Predictor en tiempo real de patrones armónicos

Trabajo de Fin de Grado

Pablo Carra García
NIA: 100282668

Grado en Ingeniería Informática

Octubre 2015

Tutor:
Tomás de la Rosa Turbides

Co-Tutor:
Jesús Virseda Jerez

CONTENIDO

ÍNDICE DE ILUSTRACIONES

1. INTRODUCCIÓN Y OBJETIVOS

1.1. Introducción

El ser humano lleva miles de años creando música. Desde sus inicios más primitivos, cuando los músicos sólo empleaban su voz y su cuerpo para crear ritmos y melodías, la música ha ido creciendo inexorablemente en complejidad y riqueza, mientras los músicos ideaban estructuras y motivos cada vez más intrincados. Aun así, la música occidental ha florecido soportada por un armatoste muy concreto: las reglas de la armonía. Tan inventadas como descubiertas, éstas reglas permean toda nuestra producción musical, dictando su desarrollo, incluso el de aquellas obras que se esfuerzan deliberadamente en romperlas. Pero, ¿hasta qué punto constriñen la composición, forzándola a quedarse en ciertos cauces? Dicho de otra manera: ¿Puede predecirse el devenir de una obra a partir de la estructura armónica que va mostrando? ¿Podría una máquina entrenada "escuchando" ciertas obras predecir acertadamente lo que va a ocurrir en una que no conoce?

Éste trabajo pretende desarrollar un sistema que sea capaz de predecir el desarrollo armónico futuro de una obra musical a partir de la progresión que ha llevado hasta un cierto punto en ella. En el ámbito de este proyecto se emplearán partituras, aunque el objetivo a largo plazo sería realizar análisis de audio real. Para extraer la información relevante de la partitura se empleará conocimiento experto, y ésta se introducirá en un modelo de predicción. El enfoque del predictor desarrollado en éste trabajo será eminentemente probabilístico, y basado en el entrenamiento realizado con una base de datos.

El trabajo se dividirá en las siguientes fases:

- Identificación de las informaciones relevantes a extraer de una partitura
- Desarrollo de un lector que extraiga las mismas
- Evaluación de diferentes modelos de predicción y elección del más adecuado
- Implementación del predictor en tiempo real empleando los componentes desarrollados anteriormente.
- Desarrollo de una interfaz de usuario

1.2. Objetivos

El objetivo final del proyecto será la implementación de un predictor armónico funcional. Para llegar hasta ese punto será necesario cumplir los siguientes hitos:

Hito 1: Elección del sistema de representación
El primer hito será la elección del sistema de representación de música óptimo para este proyecto. Dadas las circunstancias bajo las que se ejecutará la aplicación y las necesidades en cuanto al entrenamiento, el formato elegido será MusicXML.

Hito 2: Extracción de información de MusicXML
Habrá que crear un lector de ficheros que sea capaz de extraer de un fichero MusicXML toda la información que se considere necesaria. Principalmente serán las

notas que suenan, pero también deberá poder devolver información relacionada con la tonalidad, el tempo, o cualquier otro factor que pueda resultar ser de interés.

Hito 3: Preparación de la información para su tratamiento
Después habrá que implementar un sistema que convierta la información sin tratar extraída del XML a un archivo que el programa WEKA pueda emplear para entrenar modelos. Será en este punto donde la información concreta extraída de la partitura será abstraída para dar lugar a un sistema más robusto y potente.

Hito 4: Recopilación de una base de datos
Será necesario crear una base de partituras en formato MusicXML suficiente para entrenar los modelos. Ésta deberá estar adaptada al sistema en cuanto a instrumentación (partituras corales) y estilo (obras del clasicismo).

Hito 5: Elección del modelo de predicción
El quinto hito será generar distintos modelos con WEKA a partir de los datos recogidos, y posteriormente evaluarlos para elegir el más apropiado. El modelo elegido será el que emplee el predictor final.

Hito 6: Integración del modelo de predicción al sistema
Posteriormente se introducirá el modelo de predicción generado con WEKA en la aplicación, implementando las estructuras de datos necesarias para suministrar la información al modelo y configurando también qué hará el sistema con las predicciones que se vayan generando.

Hito 7: Implementación de una interfaz de usuario
El último paso será crear una interfaz de usuario que permita elegir cómodamente qué archivo analizar, y que muestre los resultados del sistema de una manera gráfica y en tiempo real.

1.3. Estructura de la memoria

A continuación se presentan resúmenes breves de los capítulos de este trabajo a modo de guía.

Capítulo 1: Introducción y objetivos. Introducción al proyecto y explicación de los objetivos que se intentarán cumplir en él.

Capítulo 2: Teoría musical. Breve resumen de los conceptos de teoría musical básicos que se usarán en el proyecto.

Capítulo 3: Estado del arte y herramientas empleadas. Explicación de la situación actual de la investigación en este campo y descripción de las herramientas que se emplearán en el proyecto.

Capítulo 4: Definición y desarrollo del problema. Definición de los problemas a resolver en profundidad, y explicación del planteamiento que se empleará para resolverlos.

Capítulo 5: Implementación. Se proporcionará una perspectiva de la implementación realizada, profundizando en los aspectos más importantes.

Capítulo 6: Planificación y presupuesto. Presentación del plan de desarrollo de software diseñado, y presupuesto del proyecto.

Capítulo 7: Pruebas. Descripción de las pruebas realizadas y discusión de los resultados de las mismas.

Capítulo 8: Marco regulador y entorno socio-económico. Explicación del marco regulador que rodea el proyecto e imagen del entorno socio-económico en el que se sitúa.

Capítulo 9: Conclusiones y perspectivas de desarrollo. Revisión de los resultado obtenidos a lo largo del proyecto, y discusión de vías de desarrollo futuro.

Capítulo 10: Referencias. Índice de las referencias empleadas durante el proyecto.

Capítulo 11: Anexos. Aquí se incluirá información que es relevante para el proyecto, pero cuya extensión entorpecería la lectura.

Capítulo 12: Summary in English. Resumen en inglés del proyecto.

2. TEORÍA MUSICAL

En éste apartado se explicarán los conceptos musicales básicos que se emplean en este proyecto para facilitar la comprensión del mismo.

2.1. Elementos básicos: Notas y escalas

Una **nota** es el sonido que genera una vibración con frecuencia fundamental constante. Varias notas consecutivas constituyen una **melodía**, mientras una sucesión de notas simultáneas genera una **armonía**. El sistema temperado occidental considera doce notas distintas, separadas por semitonos. De estas 12 notas 7 tienen nombres simples: Do-Re-Mi-Fa-Sol-La-Si; mientras que las otras cinco reciben un nombre compuesto en relación a su situación con respecto a las otras 7 y el contexto armónico. Así, a una de esas notas se la puede denominar Do# ("Do sostenido", medio tono por encima de Do) o Re*b* ("Re bemol", medio tono por debajo de Re) según dicten las circunstancias, refiriéndose al mismo tono[1]. Una buena visualización de las notas la ofrece el piano: las teclas blancas corresponden a las notas de nombre simple, y las negras a las de nombre compuesto.

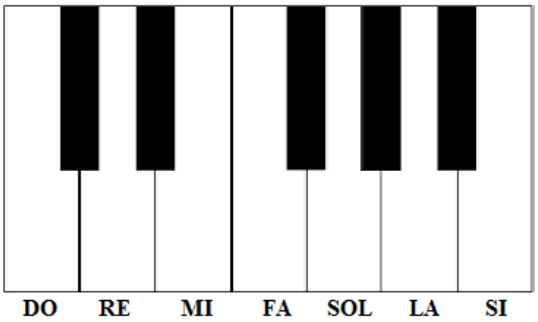


Ilustración 1: Notas del piano

Una **escala** es un grupo de 7 notas separadas por una combinación concreta de tonos y semitonos. Este conjunto (y su distribución concreta de pasos) es el que define la **tonalidad** de una obra, el ámbito armónico en el que se moverá. A partir de la tonalidad se deducen los grados tonales y sus funciones armónicas (explicadas más adelante) [1].

[1] Nótese que esta situación también se traduce a la notación musical: dos notas que en el pentagrama parecen distintas pueden describir el mismo tono. Esto se denomina **enarmonía.**

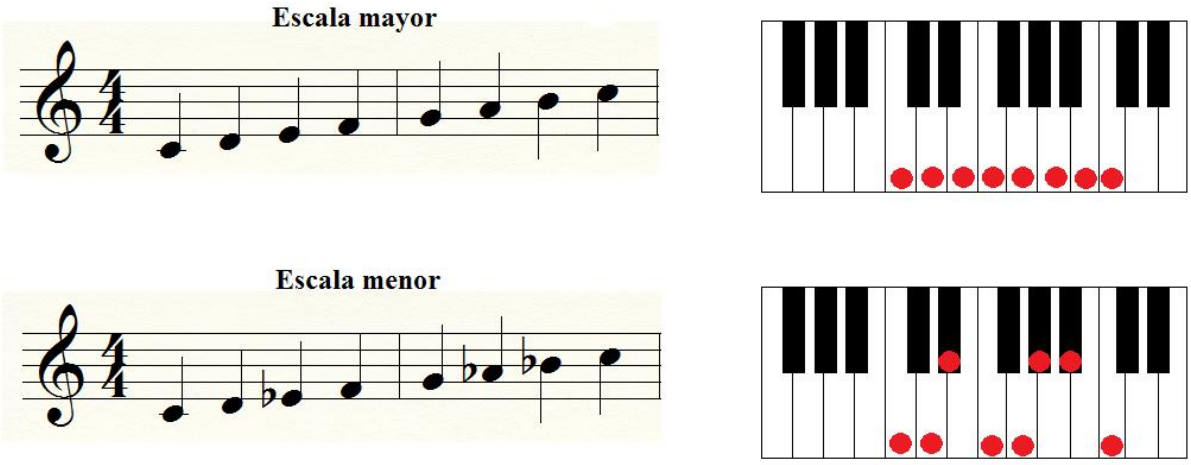


Ilustración 2: Escalas

Las escalas más comunes un nuestra música son la escala mayor (también llamada Jónica) y menor (también llamada Eólica), aunque existen otras. La escala mayor está compuesta por 7 tonos distintos (más la octava), separados todos por un tono entero salvo del tercero al cuarto grado (de la tercera a la cuarta nota) y después del séptimo, donde el paso es de medio tono. En las escalas menores, en cambio, estos pasos de medio tono se encuentran entre el segundo y tercer grado y entre el quinto y el sexto.

2.1.1. Relativos mayores y menores

Cada tonalidad mayor "convive" con una tonalidad menor, al igual que cada tonalidad menor convive con una mayor. A estas tonalidades se las llama **tonalidades relativas**.

Esta convivencia surge de la estructura de tonos y semitonos que define la escala de la tonalidad. Si empezamos a leer una tonalidad mayor por su sexto grado, la estructura de semitonos y tonos que surge es la de una tonalidad menor.

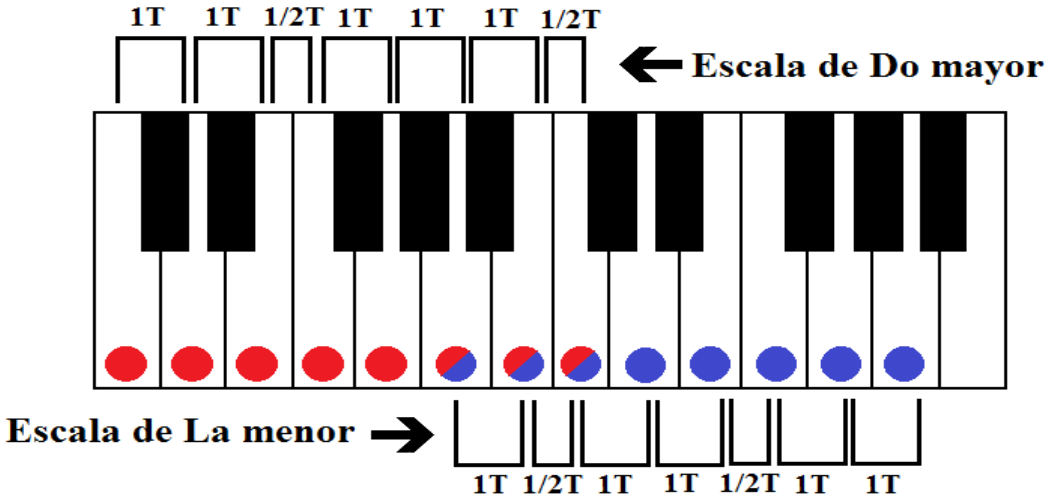


Ilustración 3: Escalas relativas

Es muy común en una obra pasar de una tonalidad a su relativo a través de cambios en la melodía y la armonía. A esto se le llama **modulación.**

2.2. Acordes

Un **acorde** es un conjunto de tres o más notas que suenan simultáneamente, formando una unidad armónica. El acorde más sencillo es la tríada mayor, que consiste en la nota que da nombre al acorde (llamada **fundamental**), su quinta (situada tres tonos y medio por encima) y su tercera mayor, situada dos tonos por encima. El siguiente acorde más importante, la tríada menor, es similar al mayor, pero con una tercera menor en vez de una mayor, un tono y medio por encima de la fundamental. También existen las tríadas aumentadas y disminuidas: una aumentada es un acorde mayor con la quinta aumentada medio tono, y un acorde disminuido es una acorde menor con la quinta disminuida medio tono.

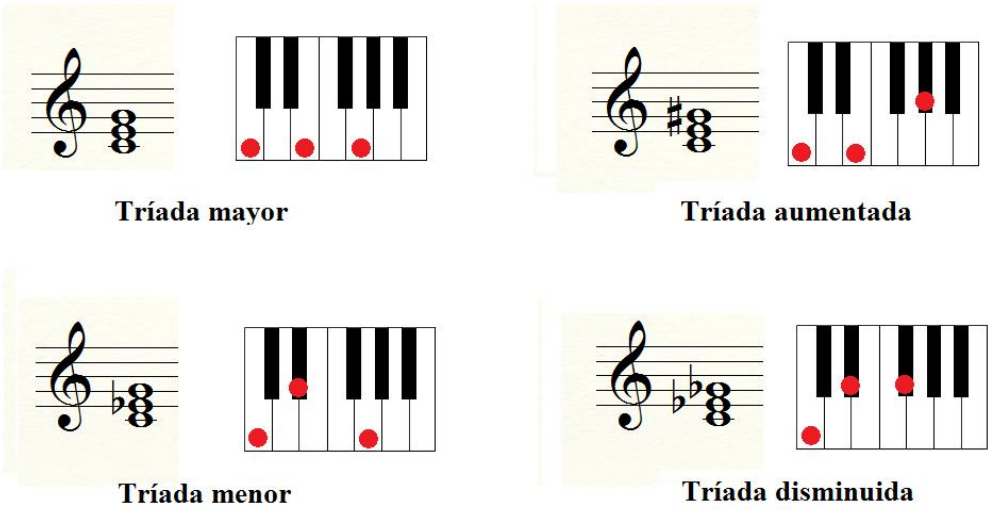


Ilustración 4: Tríadas

Más allá de las tríadas existen acordes con un mayor número de notas. Lo más común es que la cuarta nota añadida sea la séptima de la escala, mayor o menor. Es decir, la nota que está 5 tonos o 5 tonos y medio por encima de la fundamental.

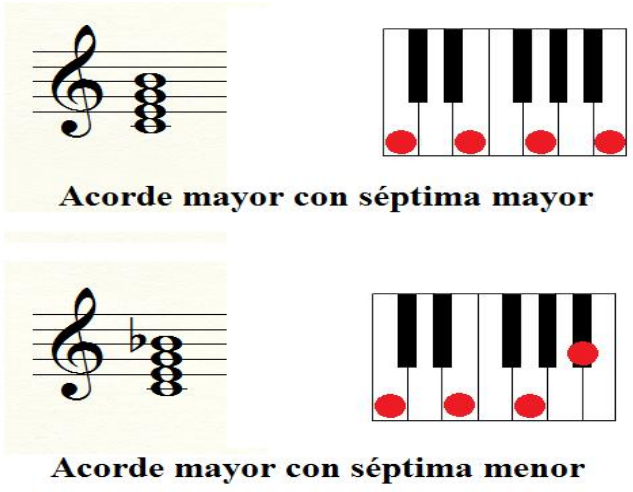


Ilustración 5: Acordes con séptima

Conviene destacar que el orden en el que suena las notas de un acorde no cambia el nombre del mismo. El proceso de cambiar las notas de orden se llama **invertir** el acorde. Así, un acorde en primera inversión tendrá como nota más grave la tercera, y la fundamental se encontrará en un registro más agudo. La función tonal de los acordes invertidos es idéntica a la de sus versiones originales, aunque a veces la inversión pueda proporcionar información adicional útil.

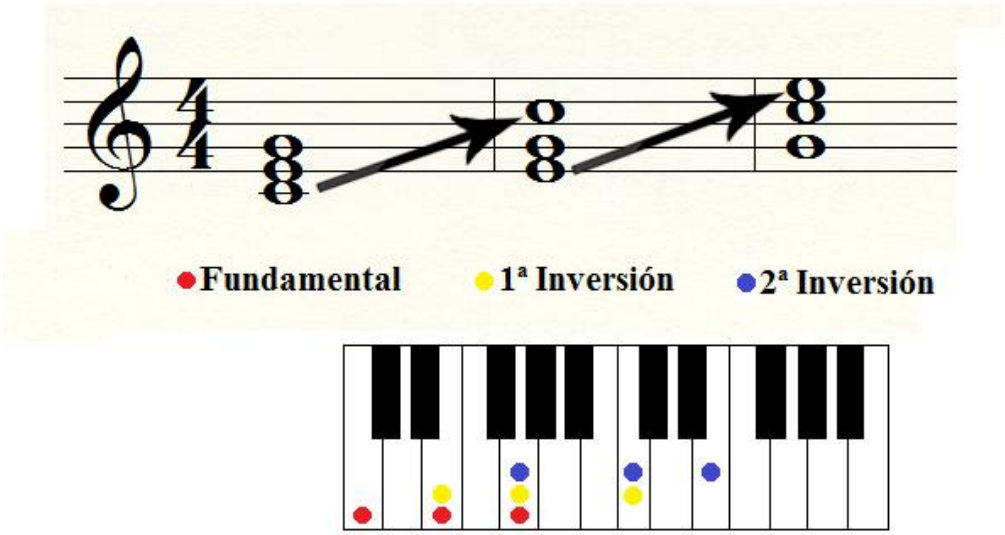


Ilustración 6: Inversiones

2.3. Funciones tonales

En cada tonalidad, a los acordes que aparecen en ella se les asigna un grado, según la posición de la fundamental en la misma. Así, al acorde cuya fundamental da nombre a la tonalidad le corresponderá el primer grado, al siguiente el segundo, etc. Cada grado en una tonalidad tiene una **función tonal**, basada en sus características sonoras [2]. Estas son:

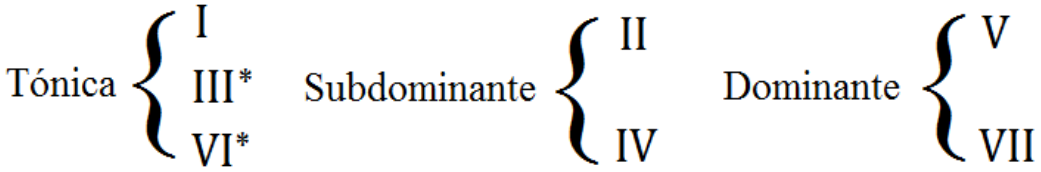


Ilustración 7: Funciones tonales

Tónica: La base de la tonalidad. Ofrece la mayor estabilidad armónica y crea la sensación de reposo en la música. El grado con ésta función es el I.

Sobre el III y VI grado: Estos grados se denominan **grados modales.** Al ser de una sonoridad más débil, no poseen una función armónica determinada, y

suelen emplearse para añadir color a la armonía, generalmente en conjunción con la tónica (lo que lleva a algunas teorías de armonía a considerarlos sustitutos del grado I). Esto se debe a que están relacionados con la tonalidad relativa de aquella en la que se encuentran: en una escala mayor, el relativo menor comienza en el sexto grado (Do mayor → La menor), mientras que en una menor el relativo mayor comienza en el tercero (La menor → Do mayor). De todas formas, el uso intensivo del III y VI grado puede confundir al oído y hacerle creer que se ha modulado a la tonalidad en la que éstos acordes corresponden a grados "fuertes": Así, una progresión que en Do mayor corresponde a los grados VI – III- VI, vista desde La menor puede ser interpretada como una progresión I – V - I, cadencia que al oído resulta más obvia. En pos de la simplicidad, y ya que no añaden la tensión que introducen las otras funciones tonales, se han tratado como Tónicas en el ámbito de este trabajo, pero conviene tener en mente sus características especiales.

Dominante: La función "opuesta" a la tónica. Es la que mayor tensión genera y suena muy inestable. Tiende a estar seguida por un acorde de tónica para resolver la inestabilidad. Está interpretada por el V grado, aunque como sustituto puede emplearse el VII.

Subdominante: Un punto intermedio. También genera tensión, pero en menor grado que la dominante. El grado IV es el que desempeña esta función, aunque también puede hacerlo el II.

El análisis de las funciones tonales de los acordes de cada compás ayuda a interpretar el ir y venir de una obra, y a comprender mejor qué está haciendo el compositor.

3. ESTADO DEL ARTE Y HERRAMIENTAS EMPLEADAS

3.1. Estado del arte

La unión de informática y música es un campo donde en la actualidad se están desarrollando diversos proyectos, y que está trayendo numerosos descubrimientos. Estos se pueden dividir en tres grandes ramas:

Sistemas de composición automática: Sistemas cuyo objetivo es crear una obra musical nueva.

Sistemas de escucha inteligente: Sistemas que reciben como entrada algún tipo de música y trabajan con ella.

Sistemas de interpretación: Sistemas que reciben una partitura y tratan de interpretarla con expresividad similar a la de un ser humano.

A continuación se describirán en detalle todos ellos.

3.1.1. Sistemas de composición automática

Los sistemas de composición basados en algoritmos llevan existiendo desde hace mucho tiempo. De hecho los primeros (y más rudimentarios) son incluso anteriores a los ordenadores mismos. Un ejemplo es el *Musikalisches Würfelspiel*, publicado en 1792 y atribuido a Mozart [3]. Éste describía un sencillo sistema donde se lanzaban dados, y en base a los resultados se elegían pequeños fragmentos de un librito compuesto con anterioridad. Encadenando estos fragmentos manualmente se construían las obras, hasta un total de 11^{16} distintas.

El nacimiento de los ordenadores abriría las puertas a nuevos niveles de complejidad y posibilidades de éste tipo de sistemas. Ada Lovelace ya reflexionó sobre ello aún antes de que Charles Baggage terminara de diseñar su máquina analítica, precursora del ordenador:

> *"Suponiendo, por ejemplo, que las relaciones fundamentales de los tonos en el ámbito de la armonía y la composición musical fueran susceptibles a esa forma de expresión y adaptación, la máquina podría componer obras musicales elaboradas y científicas de cualquier grado de complejidad o extensión" [4]*

El primer sistema de composición ejecutado en un ordenador propiamente dicho fue desarrollado por Lejaren Hiller y Leonard Isaacson en 1955-56 en la universidad de Illinois. El fruto fue la *Illiac Suite* (1957), una obra creada con el ordenador Illiac de la universidad. El sistema empleado funcionaba de la siguiente manera: La máquina generaba una "materia prima" aleatoria, sobre la que posteriormente ejecutaba varias funciones que modificaban su contenido. Finalmente, empleaba un sistema de reglas para elegir los "mejores" resultados. Más adelante, este sistema sería la base para el desarrollo de MUSICOMP por Hiller y Robert Baker en los años 60, y con el que compusieron la *Computer Cantata*. Este

sistema transformaba las funciones ejecutadas sobre la materia prima en subrutinas independientes, de manera que el investigador tenía más libertad sobre cuáles utilizar, permitiéndole imitar estilos concretos [5].

Paralelamente a los sistemas basados en reglas, surgieron sistemas estocásticos y basados en modelos de Markov, principalmente ideados por Iannis Xenakis. De todas formas, éstos métodos (explicados en su libro *Formalized Music* de 1963) [6] estaban más bien enfocados a ayudar al compositor a encontrar nuevos sonidos en el paisaje *post-serial* de la música contemporánea.

En los últimos años también se han empezado a emplear técnicas de Inteligencia Artificial, especialmente de la programación genética, para crear nuevas composiciones. El sistema NEUROGEN, desarrollado por Paul Gibson y James Byrne en 1991 codificaba diferentes piezas en genomas artificiales y luego empleaba una red de neuronas artificiales entrenada con música "real" para evaluar la calidad de los "hijos" de cada generación [7]. Aún más recientemente ha cobrado relevancia el sistema Melomics desarrollado en la universidad de Málaga por Francisco J. Vico *et al.*[8]. Basándose en éste sistema de composición automática de base genética se han creado los ordenadores "Iamus" y "Melomics109" especializados, respectivamente, en crear obras de música clásica contemporánea y música popular. Éstas se generan en un periodo aproximado de 8 minutos y son añadidas a un repositorio con varios otros millones de obras, licenciándolas bajo CC0 (licencia de dominio público creative commons). Las obras compuestas por Iamus han sido interpretadas por humanos, e incluso se publicó un disco con sus composiciones interpretadas por la Orquesta Sinfónica de Londres.

3.1.2. Sistemas de escucha inteligente

Los sistemas de escucha inteligente emplean otro enfoque: se dedican a recibir música en algún formato y trabajar con ella. Existen numerosas variantes:

Primero, están los que reciben la entrada en forma de audio o MIDI, e intentan conectar lo que escuchan con una partitura. Así, por ejemplo, el trabajo "Following a musical performance from a partially specified score", de Bryan Pardo y William P. Birmingham, une una interpretación en formato MIDI a una *lead sheet* (versión de una partitura con información reducida, cifrados, y que no necesariamente se adhiere a la sintaxis compositiva general) a base de representar el transcurso de la interpretación como un grafo acíclico dirigido [9]. Construyendo sobre éste trabajo, los mismos autores desarrollaron "Modelling Form for On-line following of musical performances", que sigue la interpretación de la obra en una partitura convencional, usando modelos ocultos de Markov para determinar qué camino ha recorrido. Éste sistema hace el programa robusto contra fallos, pequeños saltos, y repeticiones no planeadas [10].

También es común el intento de extraer información armónica de audio físico. Existen varios intentos de determinar qué acorde está sonando en una grabación empleando (otra vez) modelos ocultos de Markov. Los resultados son buenos, aunque de momento están limitados a acordes mayores y menores [11] [12].

Una mención aparte merece el sistema *HarmTrace* de W. Bas De Haas, José Pedro Magalhaes, Frans Wiering y Remco C. Veltkamp de la universidad de Utrecht. Éste sistema hace uso de la programación funcional para realizar un análisis armónico de una secuencia de acordes. El sistema es rápido y robusto, afrontando el análisis "desde la dirección

contraria": en vez de leer los acordes e intentar extraer una frase armónicamente correcta a partir de ellos, se dedica a crear la frase armónicamente correcta que más se parezca a la secuencia introducida. Por desgracia, esto lleva a que en algunas instancias el acorde del análisis no se corresponda con el real, y su funcionamiento convierte este sistema en uno limitado a realizar análisis *a posteriori.* Aun así, el sistema está haciendo progresos, y es lo más parecido a un analizador automático exhaustivo que existe en la actualidad, ya que es capaz de trabajar con una gran diversidad de acordes distintos y funciona con una mayor base de reglas armónicas que otros sistemas [13].

3.1.3. Sistemas de interpretación

El campo de los sistemas de interpretación inteligente es un campo que ha estado creciendo recientemente, y que persigue conseguir que un ordenador sea capaz de determinar por su cuenta ciertos criterios de expresividad e interpretación en una obra hasta ahora limitados a músicos humanos. Uno de los primeros trabajos en éste campo fue realizado por M.L. Johnson y detallado en su publicación "Toward an expert system for expressive musical performance" [14]. El sistema recibía una representación numérica de unas obras (en aquel caso fugas de Bach) y tras analizarla imprimía una serie de directrices de articulación y *tempo*. Estas directrices habían sido codificadas en el sistema por expertos, y al comprobar los resultados se determinó que las indicaciones que daba el sistema coincidían en un 85-90% con las que escribió uno de los expertos como referencia.

En 1998 el investigador Josep Lluis Arcos y su equipo publicaron "Saxex: A Case-Based Reasoning System for Generating Expressive Musical Performances" [15], un sistema que aprendía realizando un análisis espectral a grabaciones de estándares de Jazz y infería a partir de ellas un sistema de transformaciones con fin expresivo. A partir de allí el sistema era capaz de emplear la información extraída y aplicar esas transformaciones a frases y obras nuevas.

Algo posterior es el sistema Director Musices descrito en "Generating Musical Performances with Director Musices", de A. Friberg *et al.* , que trabaja con archivos MIDI [16]. Basándose en trabajos anteriores de su departamento, éste sistema basado en reglas implementado en Common Lisp ofrece una interfaz gráfica que permite modificar cada una de las variables de una interpretación con el fin de crear un estilo concreto. Así, por ejemplo, fijando un conjunto determinado de parámetros puede hacer que todas las obras reproducidas sean interpretadas "tristemente" o de manera más alegre.

Recientemente se ha intentado profundizar en los aspectos que definen una interpretación "expresiva". En su trabajo "Expressivity and musical preformance: Practice strategies for pianists" [17], Alfonso Benetti entrevistó a 20 pianistas profesionales preguntándoles qué definía una interpretación expresiva y que herramientas consideraban las más importantes para llevarla a cabo. Las conclusiones presentadas fueron que la expresividad es un acto de comunicación entre el intérprete y el público, y que las herramientas más importantes para transmitir esa información son (en éste orden): articulación (la elección de qué notas y pasajes entonar), *rubato* (en qué grado se manipula el tempo en una frase), el pedal (cuánto se permite que se queden sonando las notas) y el fraseo (en qué punto de la partitura se decide terminar una "frase").

El sistema que se desarrollará en nuestro proyecto se situaría en un punto intermedio entre las dos primeras de las tres grandes ramas descritas. En esencia es, obviamente, un sistema

de escucha inteligente: recibirá una partitura y intentará extraer datos útiles de ella. Pero el objetivo de éste tratamiento de los datos es realizar una predicción, lo cual, al fin y al cabo, es una variante menor de la composición automática.

3.2. Herramientas empleadas

3.2.1. MIDI

MIDI (Musical Instrument Digital Interface)[18] es un estándar empleado para conectar y comunicar con instrumentos y ordenadores, describiendo un protocolo, una interfaz digital y unos conectores determinados. Una conexión MIDI puede transmitir hasta 16 canales de información distintos.

Es un sistema basado en eventos que especifican la nota musical, el tono, la velocidad, y otras señales de control como la dinámica o el vibrato. Empezó a tomar fuerza a partir de 1983 y a día de hoy es el estándar de manipulación musical en sistemas informáticos, siendo utilizado tanto para instrumentos como para programas de composición, sintetizadores o cualquier otro sistema que emplee música. Al ser un sistema basado en eventos se ha empleado también en otros contextos, por ejemplo como sistema de gestión de focos en una obra de teatro.

El formato MIDI podría emplearse para la reproducción de las obras y, en un futuro, posiblemente emplearse en un futuro como sistema de entrada.

3.2.2. MusicXML

MusicXML [19] es un sistema de notación basado en XML. Fue desarrollado por la empresa Recordare LLC, y pese a ser software propietario su uso es libre y sin licencia. La primera versión fue publicada en 2004, y a lo largo de los últimos años ha ganado mucha popularidad, siendo a día de hoy empleado por la mayoría de programas de edición musical.

Al basarse en XML es muy fácil trabajar con éste formato, y su amplia distribución lo vuelve idóneo como formato para los ejemplos de entrenamiento.

3.2.3. WEKA

WEKA (Waikato Environment for Knowledge Analysis) [20] es una herramienta de software para el aprendizaje automático y la minería de datos desarrollado por la Universidad de Waikato y distribuido bajo licencia libre. Debido a su calidad y su sencillez de uso es ampliamente utilizado en el campo del aprendizaje automático, conformando un estándar *de facto*. Las herramientas que ofrece WEKA serán muy útiles para éste proyecto.

3.2.4. SWING

SWING es una biblioteca gráfica del entorno estándar de Java [21], que permite construir Interfaces de manera sencilla. Al ser Java el lenguaje elegido para el proyecto, emplear SWING para la interfaz es la elección natural. Se usará Eclipse Windowbuilder, un Plug-In de Eclipse que simplifica el diseño de ventanas.

4. DEFINICIÓN Y DESARROLLO DEL PROBLEMA

Una vez estudiadas las herramientas que serán de utilidad en el desarrollo del proyecto, pasamos a describir la estructura y los requisitos del mismo.

4.1. Definición del problema

El proyecto se dividirá en dos partes: La construcción de un predictor y su implementación en una aplicación con interfaz de usuario. Cada una se describe con detalle en los apartados 4.2 y 4.3, respectivamente. Aquí se mostrará un breve resumen:

Predictor: Éste es el primer apartado a llevar a cabo en el proyecto. Se desarrollará un parser cuya tarea será leer las partituras en el formato MusicXML y extraer de ellas los datos relevantes para la predicción. Estos datos se modelarán para usarlos en WEKA, y allí se analizará cuál es el modelo de predicción más efectivo.

Interfaz y aplicación final: En esta segunda parte del proyecto se adaptará y unirá el parser creado anteriormente a una interfaz de usuario, y se integrará el modelo de predicción elegido en la misma. Éste será el producto que llegue al usuario, y le permitirá cargar un archivo sobre el que realizar las predicciones y ver los resultados de una manera sencilla e intuitiva.

4.2. Construcción del predictor

El desarrollo del predictor es un proceso de minería de datos: se extraen las características relevantes de un conjunto de datos y se emplea el conocimiento obtenido para resolver los problemas planeados. Por ello, a la hora de desarrollar el predictor, estructuraremos el trabajo usando el estándar de minería de datos CRISP-DM (Cross Industry Standard Process for Data Mining)[22] como referencia. Éste consta de 6 pasos:

1. Comprensión del negocio. Aquí se definen los objetivos a nivel empresarial, que se traducen posteriormente a un problema de minería de datos.

2. Comprensión de datos. En éste paso se observan los datos que se tratarán y se identifica qué características se usarán con el objetivo de extraer la información oculta.

3. Preparación de datos. Aquí se llevan a cabo todas las acciones destinadas a construir el conjunto total de datos sobre el que se realizarán los análisis.

4. Modelado. Esta es la fase en la que se da forma a los datos extraídos para crear los mejores modelos posibles.

5. Evaluación. Llegados a éste paso ya se ha construido el modelo, y se evalúa para determinar su calidad final.

6. Despliegue. En esta fase final se da uso al modelo creado, empleándolo para el fin que se especificó al principio del proyecto.

El modelo CRISP-DM es el más usado en el campo de la minería de datos, y se adecúa en gran medida a lo que queremos llevar a cabo. De todas formas, ciertos aspectos quedan fuera del ámbito de éste proyecto, mientras otros necesitan ser descritos con más detalle. Por ello, hemos unido los primeros dos pasos en uno ("Comprensión y definición del problema") y dividido el paso de evaluación en dos: uno, en el que se describen comparativamente los resultados de todos los modelos, y otro en el que se analiza más en profundidad el modelo ganador. A continuación se exponen los distintos pasos.

4.2.1. Comprensión y definición del problema

El desarrollo del predictor es, en esencia, un problema de clasificación: Observamos un conjunto de datos y les asignamos una clase concreta, que en éste caso sería una clase del tipo "grupos de acordes a los que les seguirá X". En nuestro caso la clase a predecir es la función armónica del acorde siguiente, es decir, si vendrá un IV grado mayor, un VII menor, etc. Para ello, deduciremos la progresión de acordes de una partitura y la guardaremos de manera estructurada, extrayendo de allí el conocimiento. Un diagrama general del discurrir del sistema tendría el siguiente aspecto:

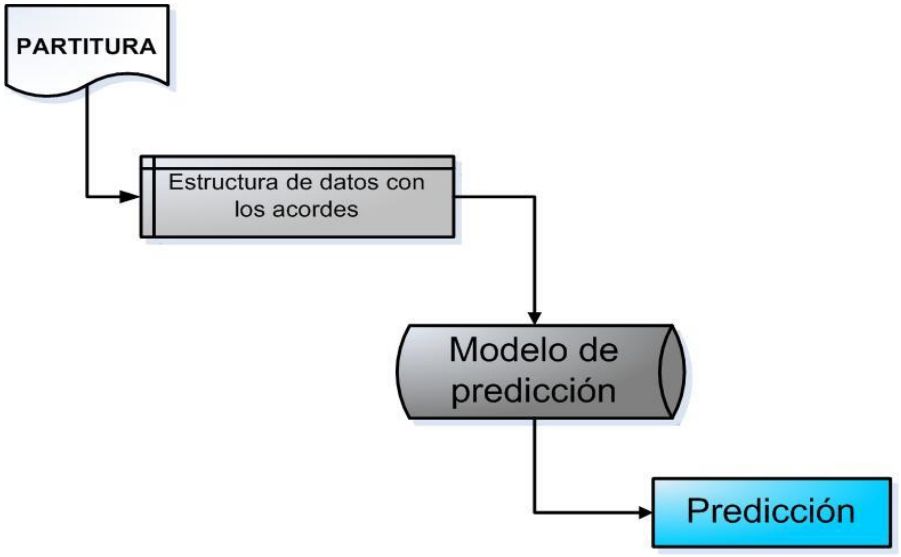


Ilustración 8: Estructura general del proceso

Para entrenar el modelo se deberá emplear una base formada por partituras completas, donde se puede leer inmediatamente la clase que sigue y por tanto comprobar fácilmente en qué grado el modelo acierta o falla. Las clases sobre las que se efectuará la predicción se extraerán de las partituras en bruto aplicando conocimiento experto y simulando una interpretación en tiempo real. En los apartados siguientes se explica en detalle el proceso de tratamiento de los datos, pero a continuación se muestra un gráfico ilustrativo de la idea general.

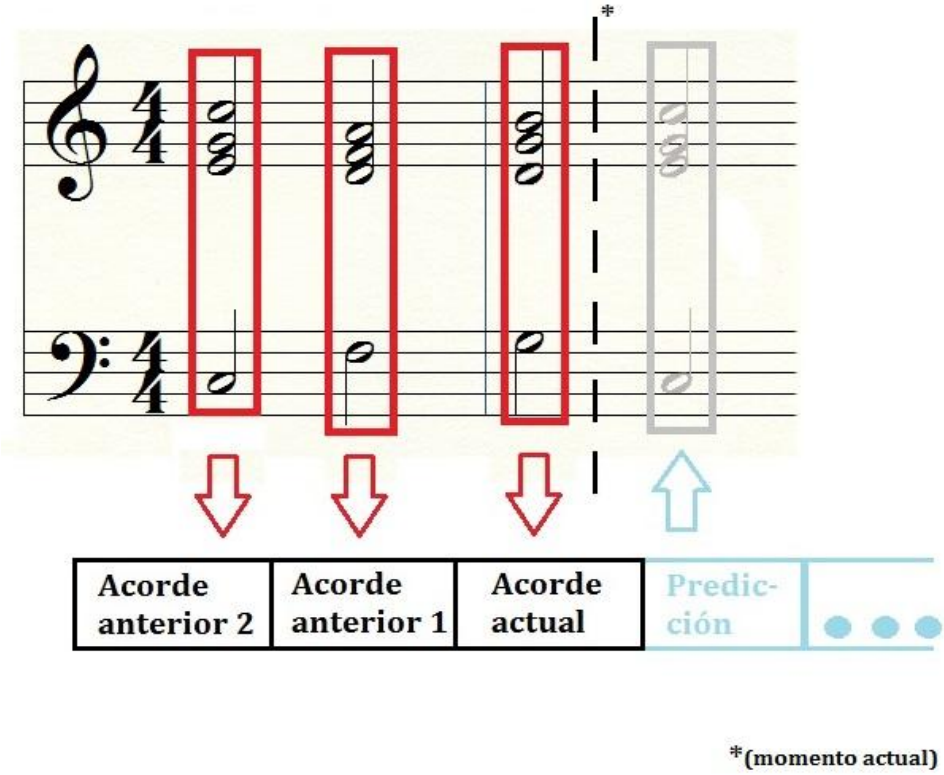


Ilustración 9: Esquema general del funcionamiento

El objetivo final sería un despliegue en forma de predictor de audio en tiempo real, pero el proyecto actual sólo aspira a ser una prueba de concepto y asentar las bases para un trabajo futuro. De todas formas, no se dejará de lado el desarrollo de una interfaz de usuario que sirva también de prototipo del uso final de la aplicación (aunque a su vez tenga en cuenta la vertiente experimental de éste proyecto mostrando estadísticas, etc).

4.2.2. Recolección de datos

Para generar un modelo de predicción útil hará falta una base de datos de la que extraer la información. En numerosos proyectos de minería de datos éste es el paso más sencillo: muchas veces ya se posee una gran cantidad de mediciones realizadas durante un período de tiempo que han sido almacenadas de manera organizada, y el reto está en encontrar la manera de darles el mejor uso. En nuestro caso, no es así. Nosotros partiremos de una base de archivos cuyos contenidos no han sido tratados previamente, y sobre los que hay que efectuar una extracción de características relevantes.

Como archivos base sobre los que efectuar la extracción emplearemos 20 obras corales distintas en formato MusicXML. Se eligieron obras del período clásico tardío de autores como Beethoven, Schubert o Haydn, ya que la música de esta época presenta las características que son de mayor utilidad para nuestro propósito: adherencia (generalmente) más estricta a las reglas de la armonía y modulaciones sencillas. En otras épocas como, por ejemplo, el barroco, la conducción de voces solía ser contrapuntísitca (especialmente en las obras de Johann Sebastian Bach), y no estrictamente armónica, y al llegar el romanticismo

con frecuencia empezaron a emplearse una gran cantidad de cromatismos y modulaciones que dificultarían nuestro análisis hasta el punto de hacerlo irrealizable.

Otra razón por la que se redujo el ámbito de la base de archivos a obras corales es que éstas suelen poseer estructuras armónicas más claras, con un número de notas simultáneas y floreos menor que, por ejemplo, las pianísticas para solistas.

Para dar uso a la información contenida en las partituras, se desarrollará un parser que será capaz de leer el archivo MusicXML y extraer la información que necesitamos. Teniendo en cuenta el sistema y su aplicación futura, esto se hará simulando una reproducción y extrayendo "momentos" concretos: cada cierto número de milisegundos se mirará que notas están sonando; esa "medición" será modelada y será la que se guarde de manera estructurada, para a partir de ella determinar el acorde, la función armónica, y las demás características que se consideren oportunas. Para poder realizar esto de una manera efectiva, el programa "montará" una versión virtual del XML usando JDOM, que permite navegar el archivo como un árbol. Esto también ayudará a extraer la información general relevante para el algoritmo antes de iniciar la reproducción de una manera sencilla, tarea que sería más difícil con sistemas que funcionan "al vuelo" como JAX.

Estructurar la lectura de esta manera también será práctico de cara a la segunda parte del proyecto: la aplicación de usuario. El parser podrá volver a emplearse sin hacer modificaciones al núcleo de su funcionamiento, y solo será necesario sustituir las clases de salida (que añaden sistemáticamente cada conjunto de datos preparado a un fichero interpretable por WEKA que luego se usará como base de datos) por otras preparadas por el uso en tiempo real de los datos obtenidos.

Sobre el "sampling"

Cabe preguntarse por qué se ha elegido un método como el que se ha descrito para descubrir que acorde suena en cada momento. Al fin y al cabo, un vistazo momentáneo a las notas que suenan en un instante concreto puede parecer poco: si resulta que en ese instante las voces se encuentran ejecutando arpegios en semicorcheas, la nota que suene en un instante exacto brindará poca información sobre el total, e incluso puede resultar engañosa si se trata de una nota de paso. Esta elección se debe a la naturaleza de la música que se pretende analizar: en la composición de obras corales siempre se tiene muy en cuenta la conducción de voces, y generalmente su movimiento es en bloque. Esto significa que la obra tenderá a quedarse periodos relativamente largos sobre una misma nota (con una ligera variación de la voz principal), lo que permite que una medida en un momento concreto pueda extraer información fiable sobre la estructura armónica en ese instante sin miedo a demasiado "ruido". Otra ventaja que ofrecería este sistema sobre otros que, por ejemplo, miraran todas las notas del compás[2], es que este sistema permite percibir progresiones dentro de un mismo compás, algo muy común. Solamente hace falta ajustar el ritmo de sampling de tal forma que "mire" varias veces por compás.

[2] Sobre el funcionamiento de este método se habla en el capítulo 9.2 "Vías de desarrollo futuro".

4.2.3. Modelado de los datos

Una vez extraída la información en bruto de la partitura, es el turno de modelarla para poder trabajar con ella en WEKA. En éste proyecto el modelado tiene dos pasos: etiquetar y abstraer.

Paso 1: Etiquetar

El objetivo de éste primer paso es definir a qué acorde corresponden las notas que están sonando. Así se obtiene por primera vez información real de la partitura y no simplemente datos: un acorde es una entidad musical propia, sujeta a unas reglas y con una función armónica específica (como se explicó en la introducción teórica). El proceso empleado para determinar cada acorde de un conjunto de notas se basa en conocimiento experto, y se detalla en el capítulo 5.2.2.

P**aso 2: Abstraer**

Finalizado el paso anterior se tiene un grupo de acordes, pero la información que aportan es muy específica: la función armónica de cada acorde depende de la tonalidad, y la progresión de acordes no nos enseña nada fuera de ella. Si leemos el acorde de Do sin saber en qué tonalidad estamos, será imposible saber si está haciendo de Tónica, de Dominante o de Subdominante, y por tanto no se podrá predecir el devenir armónico a partir de ello. Por ello, llegados a éste punto es necesario abstraer el acorde y sustituirlo por el grado que le corresponde en la escala. Así, si estamos en Do mayor, el acorde de Do pasará a ser I, si estamos en Fa lo representaremos como IV, y así sucesivamente.

Estos datos abstraídos son los que se guardarán en el fichero ARFF que empleará WEKA para generar sus modelos. Para optimizar los resultados de los modelos, a la hora de representar la información de un acorde en el fichero se optó por dividirlo en tres partes que representan atributos relevantes para el análisis del mismo. Estos atributos son:

Tríada fundamental: indica el grado al que corresponde el acorde en la tonalidad y la tríada base del mismo. Esta puede ser mayor (para acordes mayores o aumentados) o menor (para acordes menores o disminuidos). El atributo consta de 24 clases distintas: el equivalente mayor y menor para cada uno de los 12 grados posibles[3].

Modo: aquí se añade detalle sobre el modo del acorde. Se contemplan 5 tipos: una tríada "a secas", sin ninguna información adicional (representada por el símbolo "="), una triada con séptima mayor o menor (que añade tensiones) o un acorde aumentado o disminuido.

Bajo: generalmente a la hora de conducir voces, el bajo suele brindar información sobre a dónde se va a dirigir la armonía: por ejemplo, si está sonando el primer grado mayor (pongamos que Do mayor), el acorde sonará distinto si la nota más grave que suena es un Do (la fundamental) que cuando es un Sol (la dominante), incluso aunque ambas notas sean parte natural del acorde. Si el bajo está en Sol, la sensación será

[3] En la introducción teórica se enseñó que una escala tiene 7 grados naturales. Esto es correcto, pero hay que tener en cuenta que el sistema tradicional comprende 12 tonos distintos. Estos 5 grados que se encuentran fuera de la tonalidad se cifran como "versiones rebajadas" de los que les siguen. Así, el acorde entre el primer y segundo grados se cifra como bII (segundo grado bemol).

más inestable, y probablemente la música vaya a seguir progresando hasta algún punto, mientras que si el bajo está en Do, la sensación será mucho más "completa" y la probabilidad de que se haya acabado una frase es mucho más alta. En éste atributo contemplamos 12 clases, que indican el grado de la nota más grave de las observadas.

Acordes: {IM, Im, bIIM, bIIm, IIM, IIm, bIIIM, bIIIm, IIIM, IIIm, IVM, IVm, bVM, bVm, VM, Vm, bVIM, bVIm, VIM, VIm, bVIIM, bVIIm, VIIM, VIIm }

Modos: {=, 7M, 7m, dim, aug}

Bajo: {I, bII, II, bIII, III, IV, bV, V, bVI, VI, bVII, VII}

Ilustración 10: Diagrama resumen de las clases de los atributos

Para realizar las predicciones se ha decidido tener en cuenta el acorde actual y los cuatro anteriores. Éste total de 5 acordes se empleará para predecir la tríada fundamental del acorde siguiente. En cualquier caso, la base de datos guardará también el modo y el bajo de los acordes siguientes en los ejemplos de entrenamiento, por si se necesitara cambiar la clase a predecir. En consecuencia, una entrada en la base de datos de entrenamiento tendrá el siguiente aspecto:

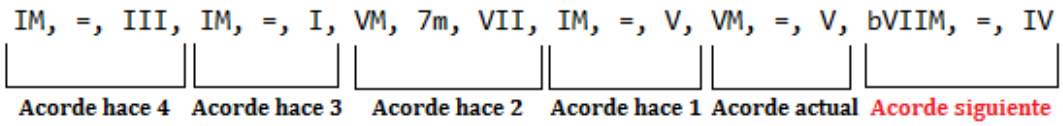


Ilustración 11: Ejemplo de una entrada de la base de datos

4.2.4. Generación y evaluación de los modelos

Para analizar la información extraída de las partituras y generar el motor de predicción se probarán diversos clasificadores disponibles en WEKA. Todos los que se emplearán están disponibles en la versión básica, salvo los modelos LVQ y SOM, que forman parte del Plug-In "WEKA Classification Algorithms" [23]. Se probarán los siguientes tipos de clasificador:

Zero R:
Es el clasificador base. Simplemente predice la clase mayoritaria, y se usará como referencia para los demás clasificadores.

Tabla de decisión:
Un clasificador de Tablas de decisión es un modelo que genera un conjunto de reglas sencillas según las cuales decidir la clase. Es un predictor sencillo y rápido, pero que en condiciones correctas puede dar resultados a la altura de modelos más complejos [24].

BFTree:
El BFTree (Best-First-Decision Tree) [25] es un árbol de decisión que desarrolla en profundidad, expandiendo los nodos por el orden que más reduzca la impureza Gini. Ofrece la posibilidad de pre-poda y post-poda.

Árbol ID3:
El árbol ID3 es un árbol de decisión que basa el desarrollo de nodos en la ganancia de información (entropía). Está pensado para datos discretos.

Árbol J48:
El árbol J48 es una implementación Java del algoritmo de árbol **C4.5**. Está basado en el ID3, pero con las siguientes mejoras: Es capaz de trabajar con datos continuos, puede funcionar con datos a los que les faltan atributos y ofrece opciones de poda del árbol una vez terminado.

Naive Bayes:
El Naive Bayes (Clasificador Bayesiano ingenuo) es un clasificador probabilístico basado en el teorema de Bayes. Su principal simplificación es que asume que cada una de las variables es totalmente independiente y contribuye a la probabilidad final sin tener en cuenta la presencia o ausencia de otras características. Esta simplificación permite trabajar a mayor velocidad y requiere una cantidad de datos de entrenamiento menor.

Perceptrón multicapa:
El perceptrón multicapa es una red de neuronas artificial que se basa en la estructura del perceptrón, pero añadiendo capas ocultas conectadas hacia adelante, lo que permite su uso para aproximar relaciones no lineales. Ofrece una alta tolerancia a errores, y sus largos tiempos de entrenamiento se ven compensados por una gran velocidad una vez entrenado.

Mulitpass SOM:
El SOM (Self-Organizing Map) es un algoritmo de aprendizaje no supervisado basado en una red de neuronas artificiales. Las diferentes neuronas (que en éste caso representan las clases, aunque no en proporción 1:1) compiten entre sí al introducir los datos. La neurona ganadora refuerza sus conexiones y por tanto "se acerca" a ese patrón de entrada, moviendo consigo a sus vecinas directas. Se empleará la versión multipase, que funciona en dos vueltas: una primera con pocos ciclos de entrenamiento, pero con una tasa de aprendizaje y de vecindario alta, para encontrar una estructura aproximada rápidamente, y una segunda más exhaustiva, con vecindario y aprendizaje bajos, para "pulir" los resultados de la primera fase.

Multipass LVQ:
El LVQ (Learning Vector Quantization) es una modificación del SOM original. Funciona de manera supervisada, asignando de antemano a cada clase una (y sólo una) neurona. Si al introducir los datos de ejemplo la neurona ganadora es de la clase correcta, se acerca, pero si es de la clase equivocada se penaliza, alejándola. En cualquier caso, el vecindario se queda sin modificar. Aquí también se empleará la el sistema multipase, de manera similar al SOM.

Con el parser descrito anteriormente se generó a partir de las 20 obras corales una base de datos con 1568 instancias. Sobre ésta se crearon con WEKA los distintos clasificadores, fijando como objetivo de la clasificación el atributo "siguienteAcorde", es decir: la tríada

fundamental del próximo acorde. Para garantizar que los resultados no sufrieran sobreajuste, se empleó validación cruzada de 10 iteraciones en todos los modelos. A continuación se muestra una tabla resumen de los resultados de los algoritmos. La información completa puede encontrarse en el ANEXO 3.

Clasificador	Porcentaje de aciertos	Porcentaje de errores
ZeroR	22,1301%	77,8699%
Tabla de decisión	32,6531%	67,3469%
BFTree	33,4821%	66,5179
ID3	35,1403%	35,0765%
J48 (C4.5)	37,8827%	62,1173%
Naive Bayes	33,0995%	66,9005%
Perceptron multicapa	25%	75%
Multipass SOM	23,9796%	76,0204%
Multipass LVQ	16,8367%	83,1633%

Tabla 5: Resultados de los modelos

El modelo que mejores resultados dio fue el J48, mejorando los resultados del clasificador de referencia (ZeroR) en un 15%. Por tanto, fue el que se eligió para la aplicación.

4.2.5. Evaluación del modelo ganador

Como se ha visto, el modelo J48 ha sido el que ha generado los mejores resultados. Aun así el porcentaje de aciertos, pese a ser más que respetable para un clasificador de 24 clases distintas, sigue siendo bastante inferior al de error. Vamos a mirar la matriz de confusión para observar más detalladamente dónde se encuentran los errores (imagen en la página siguiente):

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	<-- classified as
212	1	8	8	1	1	6	6	3	1	25	0	55	1	2	1	2	1	0	0	1	4	7	1	a = IM
3	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	b = Im
16	0	14	2	0	0	3	1	1	0	4	0	22	0	0	0	2	1	0	0	2	0	1	0	c = IIM
22	0	2	7	0	2	2	0	0	0	9	0	7	0	2	0	2	0	0	0	1	0	4	0	d = IIm
1	0	0	1	0	2	1	0	0	1	0	0	2	0	5	0	1	0	0	1	3	0	0	0	e = bIIM
2	0	0	1	3	5	2	0	1	0	2	0	3	0	4	0	2	0	0	0	3	0	0	0	f = bIIm
13	0	3	5	1	3	7	0	4	0	5	0	12	0	1	1	5	1	0	0	10	0	1	0	g = IIIM
10	0	1	0	0	0	0	3	0	0	3	0	5	0	1	0	2	0	0	0	0	1	0	1	h = IIIm
5	0	0	0	0	0	5	1	10	0	2	0	6	0	1	0	0	0	0	0	9	0	0	1	i = bIIIM
0	0	0	0	0	0	2	2	0	4	0	0	2	0	3	0	1	0	0	0	2	0	0	1	j = bIIIm
49	0	2	5	1	0	2	0	1	0	50	0	16	0	0	0	3	6	0	0	0	0	2	0	k = IVM
3	0	0	0	0	0	1	0	0	1	0	0	2	0	0	0	0	0	0	0	1	0	1	0	l = IVm
74	0	16	3	0	0	5	0	6	0	12	1	162	0	3	0	9	5	0	0	5	2	2	0	m = VM
5	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	n = Vm
6	0	1	4	2	4	3	0	3	3	1	0	2	0	20	1	1	1	0	1	16	0	2	2	o = bVM
2	0	0	0	2	0	0	1	0	0	0	0	0	0	0	6	0	0	0	0	1	0	0	0	p = bVm
12	0	3	1	1	3	3	3	1	1	7	0	18	0	4	0	17	1	0	0	3	0	1	0	q = VIM
10	0	2	1	0	1	1	1	1	0	6	0	12	0	1	0	1	5	0	0	0	1	0	0	r = VIm
3	0	0	1	2	3	1	0	5	0	0	0	5	0	0	0	1	0	3	0	0	0	3	0	s = bVIM
2	0	0	0	1	1	1	0	3	2	0	1	0	1	1	0	0	0	0	0	2	0	0	1	t = bVIm
4	0	0	3	5	5	4	0	7	0	2	2	4	0	7	0	1	1	1	1	41	0	0	0	u = VIIM
7	0	0	1	0	1	0	1	0	0	0	0	4	0	0	0	0	1	0	0	0	4	1	0	v = VIIm
14	0	1	3	0	0	0	0	1	0	3	0	4	0	1	0	3	0	0	0	1	1	24	0	w = bVIIM
2	0	0	0	0	0	1	1	0	1	0	0	0	0	4	0	0	0	0	0	0	0	0	0	x = bVIIm

Ilustración 12: Matriz de confusión del algoritmo J48 (se puede encontrar una imagen mayor en el anexo)

Como se puede ver en las líneas 1 y 13, el error más común es que el sistema prediga un primer grado mayor cuando llega un quinto y viceversa. Aunque a primera vista pueda parecer contra intuitivo, esta confusión es razonable: el primer grado y el quinto son los más alejados armónicamente, y el empleo de uno cuando se espera el otro suele ser una acción deliberada del compositor para, repentinamente, crear tensión (en el caso de "introducir" un quinto grado) o forzar una resolución de la frase musical (llevando el progreso de vuelta a un primer grado). Si el sistema hubiera estado prediciendo quintos grados menores rebajados medio tono el problema habría sido más grave. Algo similar ocurre con el cuarto grado: suele marcar el inicio de cadencias o progresiones nuevas. Es decir: pese a encontrarse completamente dentro de la armonía, suele ser indicativo de un cambio, y por tanto más difícil de predecir. Mitigar los errores de clasificación en estos momentos críticos aunque especialmente problemáticos de predecir es una de las vías de desarrollo futuro planteadas en las conclusiones.

4.2.6. Despliegue

El último paso de la metodología CRISP es el despliegue. Aquí es donde se define en qué manera se dará uso al conocimiento obtenido en los pasos anteriores. A veces esto se limita a una representación estructurada de los datos obtenidos para informar al lector y servir de referencia. En nuestro caso el objetivo es crear un sistema que asista en la interpretación real, por lo que el despliegue consistirá en desarrollar una aplicación de usuario con interfaz gráfica que emplee el modelo de predicción creado. Éste proceso es la segunda parte de éste trabajo, y su envergadura y complejidad justifican un apartado propio, que sigue a continuación.

4.3. Desarrollo de la interfaz de usuario

Una vez finalizado el predictor, es la hora de desarrollar la interfaz que se usará. Usando como base el parser y añadiendo el modelo de predicción, será necesario implementar una interfaz gráfica que permita manejarlo en tiempo real. Ésta aplicación de usuario estará sujeta a las exigencias de un proceso de desarrollo de software estándar, y por tanto deberá cumplir ciertos requisitos y ser capaz de responder satisfactoriamente ante los casos de uso más comunes de la aplicación. En éste apartado se detallará a qué deberá atenerse la aplicación final.

4.3.1. Requisitos del sistema

Como es común, los requisitos impuestos al sistema se dividirán en dos tipos:

-**Funcionales**: Aquellos que el sistema debe proporcionar

-**No funcionales**: Las condiciones que debe cumplir el sistema

A continuación se mostrará una tabla resumen de los requisitos definidos, junto con la información más importante de cada uno. La especificación completa se puede encontrar en el ANEXO I.

Requisitos funcionales:

Referencia	Título	Descripción
RF-01	Menú	La aplicación debe tener un menú en forma de interfaz gráfica.
RF-02	Selector de archivos	La aplicación debe permitir elegir un archivo a través de una ventana de buscador de archivos.
RF-03	Play	La aplicación debe tener un botón que permita iniciar la reproducción del archivo.
RF-04	Stop	La aplicación debe tener un botón que permita detener la reproducción en cualquier momento.
RF-05	Estadísticas	La aplicación debe mostrar las estadísticas de acierto/error de la ejecución actual.
RF-06	Feedback Visual	La aplicación debe representar el estado actual de la reproducción y las previsiones de una manera clara e intuitiva.

Tabla 1: Requisitos funcionales

Requisitos no funcionales:

Referencia	Título	Descripción
RNF-01	Portabilidad	La aplicación debe poder ejecutarse en diferentes sistemas operativos.
RNF-02	Usabilidad	La aplicación debe ser usable por un usuario sin grandes conocimientos técnicos.

RNF-03	Fichero ejecutable	La aplicación debe estar contenida en un fichero ejecutable.

Tabla 2: Requisitos no funcionales

4.3.2. Casos de uso

Los casos de uso definen las acciones que el usuario podrá realizar con la aplicación, y ante los que ésta deberá ser capaz de responder. Los casos de uso desarrollados se muestran a continuación en versión resumida, como en el apartado anterior. La versión completa de los casos se encuentra en el ANEXO 2.

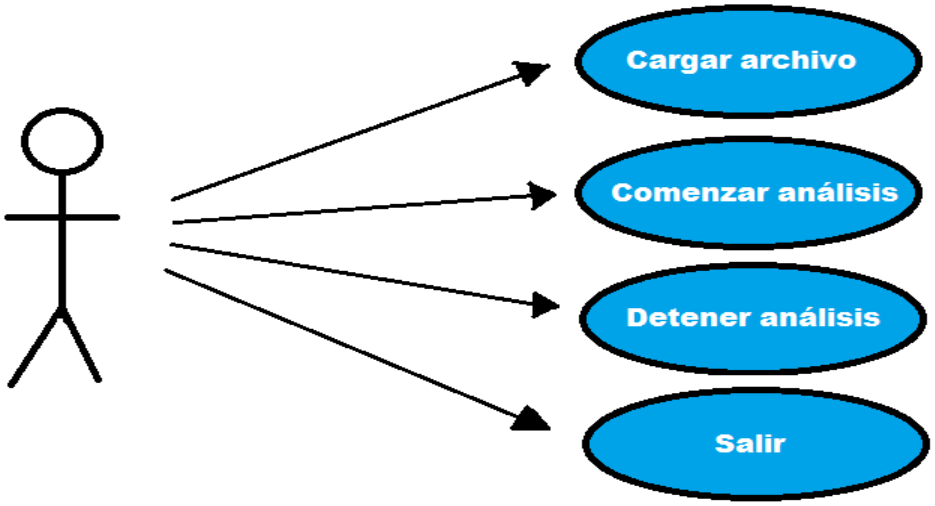


Ilustración 13: Casos de uso

Identificador	Título	Descripción
CU-01	Cargar archivo	El usuario decide sobre qué archivo ejecutar la predicción
CU-02	Comenzar análisis	El usuario inicia el análisis
CU-03	Detener análisis	El usuario detiene el análisis
CU-04	Salir	El usuario sale de la aplicación

Tabla 3: Casos de uso

Relación entre los casos de uso y los requisitos funcionales

Cada caso de uso está interrelacionado con uno o varios de los requisitos funcionales del sistema. A continuación se muestra una tabla donde se puede ver la relación entre los casos y los requisitos de ésta aplicación.

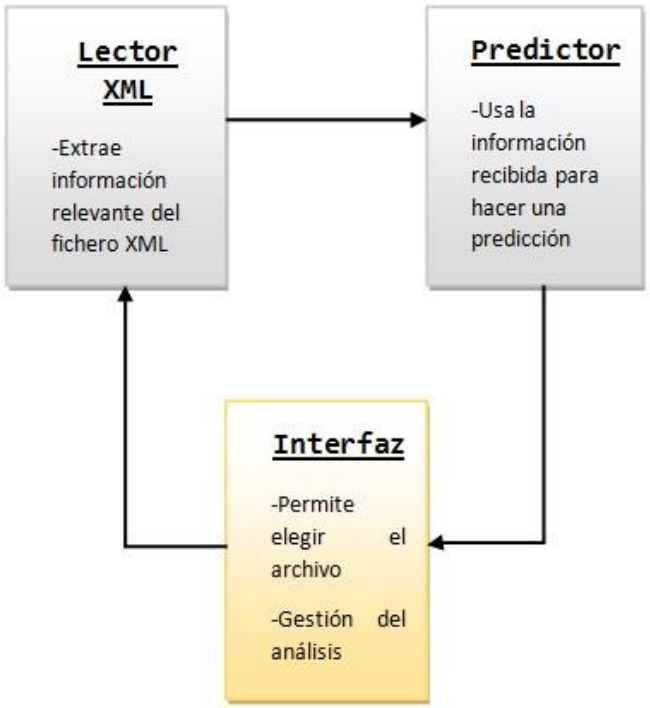
	CU-01	CU-02	CU-03	CU-04
RF-01	**X**	**X**	**X**	**X**
RF-02	**X**			

RF-03	**X**	
RF-04		**X**
RF-05	**X**	
RF-06	**X**	**X**

Tabla 4: Relación entre los casos de uso y los requisitos funcionales

4.3.3. Diagrama general

El diagrama general de la aplicación tendría la siguiente forma:



4.3.4. Características y diseño de la interfaz

La interfaz deberá ser sencilla e intuitiva. Ya que las opciones del usuario serán limitadas, no harán falta varias pantallas, con mostrar todo en una será suficiente. De todas formas, será necesario que muestre al menos:

- El nombre del archivo que se está analizando
- Botones de inicio y detención
- El acorde anterior, el actual y la predicción
- Un testigo que indique de manera rápida y visual si la última predicción ha sido correcta
- Las estadísticas relevantes del análisis actual

Atendiendo a las características descritas se ha diseñado la siguiente interfaz preliminar:

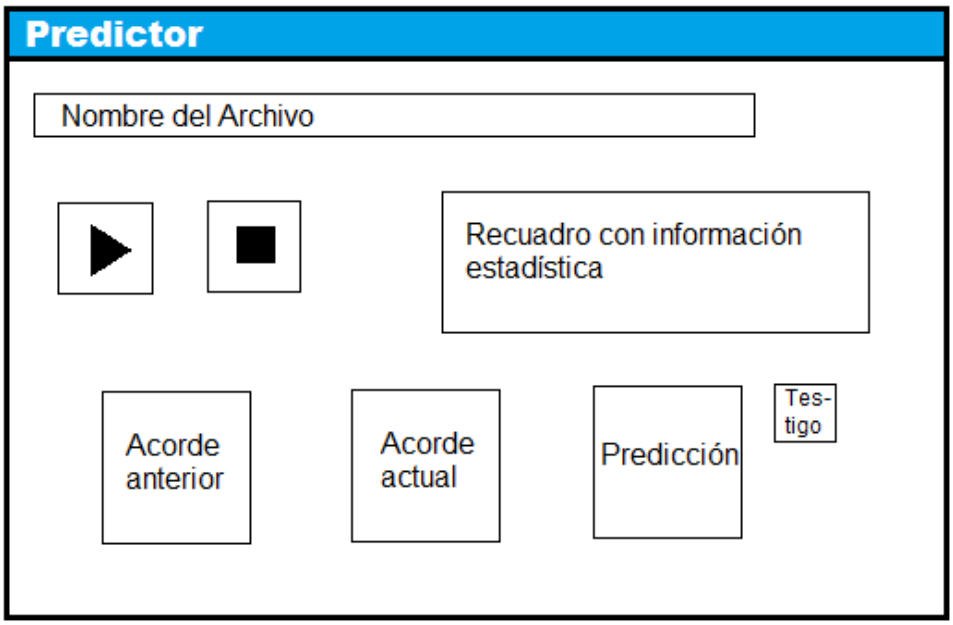


Ilustración 14: Ilustración preliminar de la interfaz

4.3.5. Herramientas para el desarrollo de la interfaz

Para el desarrollo de la interfaz se empleará la biblioteca de Java SWING, unida a la herramienta de Eclipse "Eclipse Window Builder". La sencillez de uso de estas herramientas permitirá un desarrollo rápido y efectivo de la interfaz del programa.

5. IMPLEMENTACIÓN

Una vez definido el problema, pasaremos a la implementación. En éste apartado se describirá la estructura y el funcionamiento detallado de las clases del proyecto. Se han creado diagramas de clases tanto para el total del proyecto como para cada una de las partes, pero no se muestran en este apartado para no incomodar la lectura. Pueden encontrarse en el ANEXO V.

5.1. Clase principal

Durante el desarrollo del proyecto se han creado dos sistemas de salida para el núcleo del programa: uno, para generar los conjuntos de entrenamiento del modelo a partir de partituras, y otro, para leerlas y usar un modelo para hacer predicciones. Aunque el generador de conjuntos de entrenamiento no se emplee en la versión final, también se describirá en este apartado y las clases del mismo se incluyen en el código fuente para que sirvan de referencia.

La clase InterfazPrincipal.java

La clase InterfazPrincipal es la clase principal de la aplicación. Es la que genera la interfaz y la copia en memoria de la partitura, y desde donde se gestionan todos los componentes.

Sus métodos más importantes son:

> ***Private void initialize():*** Crea la imagen de la interfaz y los *listeners* para los botones.
>
> ***Private void ejecutarLectura():*** Crea a versión en memoria del XML y un objeto Partitura para gestionarlo. Después inicia dos hilos: uno que leerá la partitura y otro que actualizará la interfaz y hará las predicciones.

5.2. Lector de XML

5.2.1. Descripción de la implementación y decisiones de diseño

El lector de XML funciona en tres pasos:

- Primero, la clase Partitura prepara en memoria la información importante de la estructura del XML e inicia una reproducción simulada (mirando que notas están sonando en un instante concreto).
- Después el conjunto de notas extraídas se envía al AsignadorAcordes. Éste asigna a través de conocimiento experto el acorde que corresponde al conjunto recibido, y devuelve el grado, el modo y el bajo.
- Finalmente, la clase Partitura envía ésta información al siguiente elemento que vaya a hacer uso de ella, ya sea el predictor o el generador de archivos ARFF.

5.2.2. Descripción en profundidad de las clases codificadas

Partitura.java

Gestiona toda la lectura. Se ocupa de la versión en memoria del árbol XML y se dedica a extraer la información importante en el momento necesario.

Métodos más importantes:

> ***Public void iniciarPartitura(List<Element> list):*** Recibe de la clase principal el conjunto de nodos que representan el árbol XML. De ahí extrae primero la información relacionada con el tempo: tanto el compás propiamente dicho, para determinar su duración en milisegundos, como las subdivisiones de cada voz, para luego poder encontrar las notas exactas en cada momento. A continuación emplea la información de la armadura para determinar la tonalidad, y crea un objeto de la clase que se va a encargar de la salida, ya sea el GeneradorAcordesParaBuffer o el GeneradorARFF.
>
> ***Public void run():*** Éste método se encarga de la lectura propiamente dicha. Funciona a través de un bucle del que no sale hasta que se haya terminado la partitura. Usando de referencia el momento en el que se comenzó la lectura, calcula la posición actual en la partitura y llama al método extraerNotas() indicando esa posición. Una vez extraídas las notas, las envía con el método enviarLista(), y a continuación el método pasa a dormir varios milisegundos antes de ejecutar otra iteración.
>
> ***Private Vector<Nota> extraerNotas(PosicionPart posicion):*** Busca en el XML la posición actual de la lectura para todas las voces, y extrae de todas ellas las notas que están sonando. Ignora apoyaturas y demás notas "falsas".
>
> ***Private void enviarLista(Vector<Nota> listaNotas):*** Recibe la lista de notas que suenan y las manda al asignador de acordes. Una vez éste la ha tratado, envía la información obtenida a la clase que se encargará de la salida, ya sea el buffer de acordes o el generador de archivos ARFF.

AsignadorAcordes.java

La función de ésta clase es decidir qué acorde corresponde a un conjunto de notas recibido.

Un acorde está definido por unos intervalos fijos entre sus notas. Por ejemplo, una tríada mayor está compuesta por su fundamental, su tercera mayor (a dos tonos por encima) y su quinta (a tres tonos y medio por encima). Estas distancias se cumplen para todos los acordes, independientemente de la fundamental en la que empiecen. Así se puede generalizar a un array de booleanos, donde cada semitono es un elemento, y los que corresponden al acorde están puestos en "true".

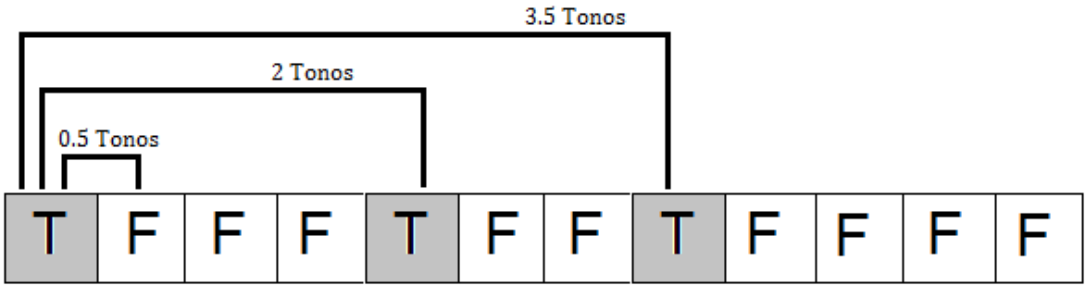


Ilustración 15: Ejemplo de la plantilla para un acorde mayor

Una vez se transforma la lista de notas a un array similar (descartando notas de duración muy reducida como "notas de paso" o floreos), se puede determinar a qué acorde corresponde el conjunto a base de compararlo con los "moldes" de los distintos tipos de acordes y sus inversiones. En caso de empate se tratará del acorde que menor cantidad total de notas precise. Así, por ejemplo, si un acorde de tríada mayor y un acorde mayor con séptima menor empatan, se trata claramente de una tríada mayor, ya que la séptima está ausente (si la séptima estuviera, el segundo acorde tendría una puntuación más alta).

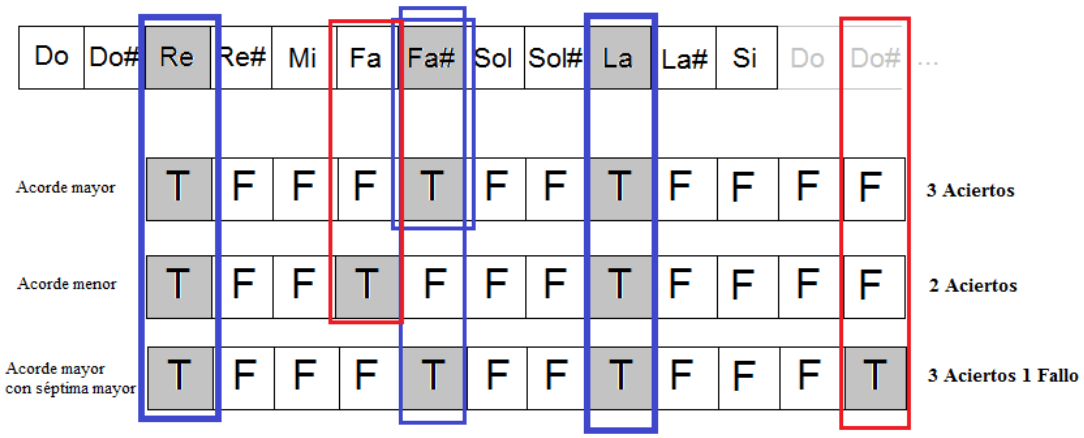


Ilustración 16: Ejemplo del sistema

Como ya se especificó en el apartado de definición del problema, los tipos de acordes que tendrá en cuenta nuestro sistema serán los siguientes: Tríadas, acordes de séptima mayor y séptima menor tanto mayores como menores, acordes aumentados y acordes disminuidos.

Métodos más importantes de la clase:

> ***Public int[] asignarAcorde(float[] original):*** Método principal de la clase. Transforma la lista de notas en un array y evalúa a qué tipo de acorde corresponde. A continuación, determina la fundamental del mismo y devuelve toda la información
>
> ***Private int[][] evaluarAcordes(boolean[] notasQueSuenan, boolean[][] estructurasAcordes):*** Este método compara el array de notas que están sonando con los diferentes "moldes" de los acordes, contabilizando las coincidencias del conjunto

recibido con los mismos. Posteriormente, se determinará qué acorde es el que suena a partir de la cantidad de coincidencias.

Public int determinarBajo(Vector<Nota> listaNotas): Este método determina la nota más baja del conjunto recibido y la devuelve en un formato adecuado para futuros procesamientos.

Nota.java

Clase que se usará para representar una nota concreta de la partitura (no un "Do" o un "La" a nivel abstracto). Contará con atributos que indiquen la octava, sus posibles alteraciones, la duración, la voz a la que pertenece, su "parte" en la obra (dato correspondiente al sistema en el que aparece) y su posición en la partitura.

PosicionPart.java

Clase que representará una posición concreta en la partitura. Poseerá dos atributos: el número de compás y el desplazo. El desplazo es un número que indica la posición normalizada dentro del mismo compás. Por ejemplo, si el desplazo es 0.5 la nota se encuentra justo en la mitad.

5.3. Predictor

En este apartado se describirá todo lo relacionado con el sistema de predicción, se describirán las clases relevantes y se indicará el modelo de predicción elegido.

5.3.1. Clases codificadas

PredictorModelo.java

Esta clase se encarga del modelo de predicción. A ella llegan los datos necesarios y los introduce en el modelo, devolviendo la predicción que éste haga.

Métodos más importantes:

Public String clasificar(Vector<Vector<String>> datos): El método que se llama desde fuera de la clase. Convierte la información recibida en una instancia de datos y la entrega al predictor. Devuelve la predicción de éste.

Public Instance crearInstanciaConDatos(Vector<Vector<String>> datos): Transforma la información del buffer en un objeto de la clase Instance, que son los que emplea WEKA. Devuelve la instancia generada.

BufferAcordes.java

Esta clase será la que guarde en tiempo real las informaciones necesarias para el funcionamiento. Poseerá tres vectores que se irán actualizando con los acordes que se van leyendo, y que darán información tanto al gestor de la interfaz como al predictor.

Métodos más importantes:

Public void anadirAcorde(String grado, String modo, String bajo): Este es el método que se encarga de añadir un acorde recién leído al buffer.

Public String[] getInfoInterfaz(): Devuelve la información que mostrará la interfaz.

Public Vector<Vector<String>> getInfoPredictor(): Devuelve todos los acordes que necesitará el predictor.

GeneradorAcordesParaBuffer.java

Esta clase se encargará de recibir la información calculada por la clase Partitura y darle el formato apropiado para poder ser usada por el buffer y el predictor. Esto consistirá en transformarla en cadenas de caracteres que se corresponden a los atributos del modelo, y en convertir los nombres "absolutos" de las notas en grados relativos a la tonalidad.

Método más importante:

Public void anadirAlBuffer(int nota, int modo, int grave): Calcula el grado relativo a la tonalidad de las notas recibidas, las convierte a String y las envía al Buffer de Acordes.

GeneradorArff.java

La tarea de esta clase es similar a la de GeneradorDeAcordesParaBuffer, sólo que en este caso la información no se guarda en el buffer que está en memoria, sino que se genera un archivo .arff que puede ser utilizado en WEKA.

Public void anadirAlArff(int nota, int modo, int grave): Inicia el proceso de conversión a grado/String, y escribe la instancia en el fichero.

5.3.2. Resultados de los diferentes algoritmos en WEKA

Se generó una base de datos con el parser y se evaluaron diferentes modelos de predicción a través de WEKA. Los detalles de estas pruebas se encuentran en el capítulo 4. Así mismo, se indicó que finalmente se optó por el algoritmo J48(C4.5), al presentar unas mejoras del 15% frente al clasificador base.

5.3.3. Matriz de costes

Ciertas predicciones pueden estar "menos equivocadas" que otras. Varios grados en la tonalidad comparten función tonal, pese a ser nominalmente distintos. Si el sistema predice correctamente un acorde que haga de dominante, aunque se haya equivocado en el acorde concreto seguirá siendo mejor que si hubiera predicho una subdominante. Para representar esto se creó una matriz de costes de error que redujera el peso de los errores "buenos".

Para ello se tuvo en cuenta:

Función tonal: Los acordes con funciones tonales similares vieron reducido su coste

Modulaciones al relativo menor: Acordes que compartieran función tonal en el relativo menor vieron reducido su coste. La información detallada sobre las tonalidades relativas se encuentra en la introducción teórica.

Se probó introducir la matriz de costes, pero ésta no mejoró los resultados generales ya que los errores no se encontraban entre los que corregía. Por ello, se decidió descartarla. Puede encontrarse en el ANEXO 4.

5.4. Interfaz

5.4.1. Versión final de la interfaz

La versión final de la interfaz no ha sufrido grandes cambios con respecto al diseño original. La única modificación consiste en que se ha añadido un botón de "Salir" en la parte inferior.

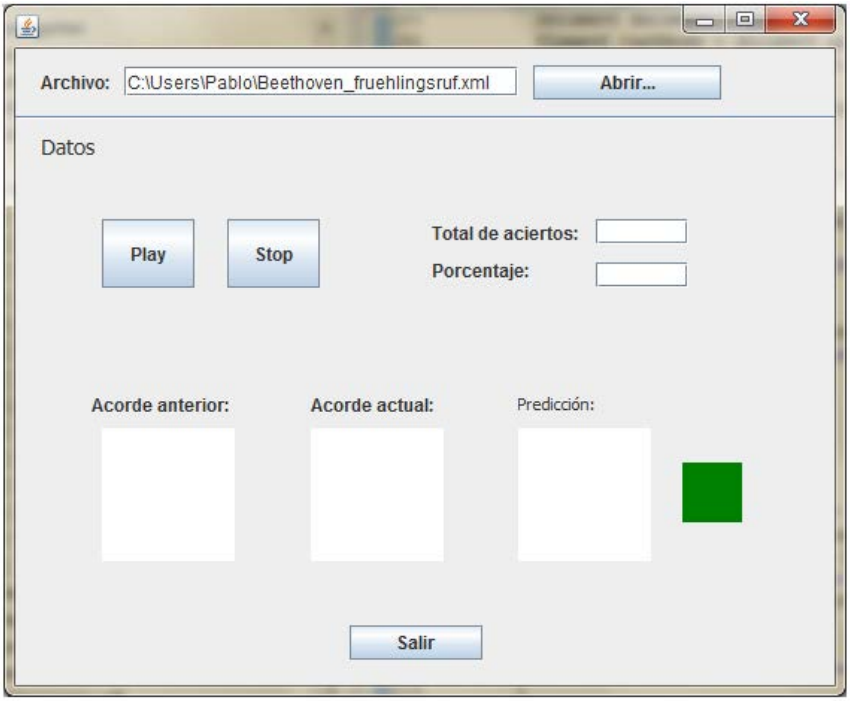


Ilustración 17: Versión final de la interfaz

Los botones son funcionales durante la reproducción del archivo, y hacer click en el botón "Abrir…" abre un explorador de archivos desde el cual seleccionar la partitura que se desea.

5.4.2. Clases implementadas

GestorInterfaz.java

Es el hilo que mantiene los paneles de la interfaz actualizados, leyendo la información de BufferAcordes y llamando al clasificador.

Métodos más importantes:

> ***Public void run():*** Método principal del hilo. Ejecuta los métodos de gestión de los paneles y estadísticas cada 100ms.
>
> ***Private void actualizarPaneles():*** Extrae los acordes actual y anterior del BufferAcordes y ejecuta el predictor, actualizando el panel de Predicción con la respuesta.

6. PRUEBAS

6.1. Tipos de baterías de pruebas

Se realizaron principalmente dos tipos de pruebas durante el desarrollo del sistema: Las pruebas para medir el funcionamiento del parser y las pruebas finales del sistema[4].

6.1.1. Pruebas al parser

El objetivo de las pruebas durante el desarrollo del parser fue comprobar que las notas que se estaban extrayendo en cada momento se correspondían con las que teóricamente estaban sonando, y más adelante verificar que los acordes asignados eran correctos. La manera más efectiva de controlar esto (sobre todo la lectura correcta) fue realizar un control manual, imprimiendo qué notas se estaban leyendo en cada punto de la partitura. La salida tuvo la forma:

```
Compas actual: 2 // Desplazo: 0.44666666
Notas actuales: [P0 C5(0.5/1)], [P0 G4(0.25/1)], [P1 E3(0.25/1)], [P1
C3(0.5/1)]
```

El control directo sobre ejemplos "reales" en vez de partituras especialmente preparadas permitió corregir numerosos errores:

Primero, ayudó a preparar el programa para gestionar eventos como tresillos, cambios de clave, o divisiones de una voz en el mismo pentagrama (y silencio de alguna de ellas), que cambian la estructura (sobre todo visual, ya que MusicXML es un lenguaje de marcado) de un compás, y por tanto podían llevar a lecturas erróneas.

Por otro lado, el uso de ejemplos reales de distintas fuentes contribuyó a crear un lector más robusto: pese a que MusicXML es un estándar con una estructura bastante fija, la información que introduce un compositor en la partitura puede ser menor a la esperada, dejando vacíos ciertos atributos. Realizar pruebas con ejemplos reales ayudó a acotar los datos necesitados al mínimo común denominador, y por tanto crear un parser preparado para trabajar con un mayor número de partituras.

Una vez revisado el funcionamiento de la lectura, se probó también el correcto funcionamiento del etiquetador de acordes. Esto se hizo primero verificando que funcionaba con ejemplos sencillos específicamente preparados, y más adelante de manera similar a la comprobación del parser: comprobando manualmente si la propuesta de acordes sobre una partitura real era correcta. Para ello se modificó la salida que ya había sido generada para el parser, añadiendo el acorde de manera que quedara como sigue:

[4] Durante el desarrollo del modelo de predicción también se realizaron pruebas a los diferentes modelos de predicción probados. Estas pruebas ya fueron descritas en el apartado 4 debido a su relevancia en el desarrollo general del proyecto. Los datos completos de todos los resultados de las pruebas se pueden encontrar en el ANEXO 3.

```
Compas actual: 2 // Desplazo: 0.44666666
Notas actuales: [P0 C5(0.5/1)], [P0 G4(0.25/1)], [P1 E3(0.25/1)], [P1
C3(0.5/1)],
Acorde: C
```

El sistema se revisó y corrigió hasta dar los resultados deseados.

A la hora de evaluar el parser deben tenerse en cuenta sus limitaciones: está diseñado para leer las notas de un momento concreto y asignar un acorde *exclusivamente a éstas.* Puede ocurrir que las notas de ese instante, debido a decisiones del compositor de motivación melódica, contrapuntística o expresiva, no sean representativas del acorde real del compás, o que éste sea sólo uno de entre los muchos que aparecen. Para minimizar este riesgo, como ya se ha explicado, se eligieron partituras corales. El objetivo de este proyecto es crear un predictor en tiempo real; la creación de un programa de análisis armónico exhaustivo de partituras es un trabajo vasto que queda fuera de su ámbito y por tanto no se acometió. Aunque sin duda sería muy bienvenido entre los alumnos de armonía de los conservatorios profesionales.[5]

6.1.2. Pruebas a la aplicación final

La evaluación final de la aplicación final también se realizó de manera manual. Primero se comprobó que el buffer funcionaba correctamente creando un registro de lo que pasa por los paneles en cada momento y comprobando que los datos son correctos: que los acordes pasan correctamente del panel "acorde actual" al panel "acorde anterior", y la predicción se actualiza a tiempo, además de que las estadísticas se generen correctamente. Teniendo en cuenta los requisitos funcionales especificados durante el diseño, el control de cumplimiento quedaría de la siguiente manera:

Requisito	Descripción	Resultado	¿Cumplido?
RF-01	Menú	La aplicación se maneja a través de un menú gráfico	**SÍ**
RF-02	Selector de archivos	La selección de archivos se realiza a través de una ventana con un selector visual	**SÍ**
RF-03	Play	La aplicación dispone de un botón que permite comenzar el análisis	**SÍ**
RF-04	Stop	La aplicación dispone de un botón que permite detener un análisis en curso	**SÍ**
RF-05	Estadísticas	La aplicación muestra las estadísticas del análisis en tiempo real	**SÍ**
RF-06	Feedback Visual	La aplicación ofrece una perspectiva del transcurso de las predicciones, junto a un testigo de referencia rápida que se muestra verde o rojo según el resultado	**SÍ**

[5] Los existentes intentos en ésta dirección han sido descritos en el apartado "Estado del arte".

Una vez comprobado el correcto funcionamiento del sistema como tal se pasó a evaluar su rendimiento, para comprobar si se ajustaba al predicho por las pruebas del modelo "en seco". Para ello, se analizaron las obras del conjunto de prueba consecutivamente y se calculó la media de aciertos según las estadísticas del programa. Los resultados fueron los siguientes:

Obra	Compositor	Aciertos	Porcentaje
Die Ehre Gottes aus der Natur	Beethoven	570/1165	48,92703862660944%
Die Himmel rühmen	Beethoven	105/253	41,50197628458498%
Flohlied	Beethoven	182/547	33,27239488117002%
Frühlingsruf	Beethoven	128/322	39,751552795031053%
Gott ist mein Lied	Beethoven	103/359	28,690807799442897%
Hymne an die Nacht	Beethoven	199/498	39,959839357429716%
Meeresstille und glückliche Fahrt	Beethoven	2906/6737	43,134926525159567%
Mignon	Beethoven	279/1073	26,001863932898417%
Nun bricht aus allen Zweigen	Beethoven	78/204	38,235294117647056%
Deutscher Walzer	Haydn	426/923	46,153846153846156%
Alles hat seine Zeit	Haydn	294/795	36,981132075471695%
An den Vetter	Haydn	607/1766	34,37146092865232%
Augenblick	Haydn	1123/2688	41,77827380952381%
Beredsamkeit	Haydn	366/1091	33,54720439963336%
Betrachtung des Todes	Haydn	214/1054	20,30360531309298%
Warnung	Haydn	539/1806	29,844961240310075%
Majestätsche Sonnenrose	Schubert	100/280	35,714285714285715%
Nächtliche Rose	Schubert	109/280	38,92857142857143%
Nächtliches Ständchen	Schubert	79/271	29,15129151291513%
24.Hymne	Schubert	559/1623	34,442390634627235%
Promedio			**36,03%**

Como se puede ver el resultado es ligeramente menor al predicho en la validación del modelo. Esto puede deberse a la varianza inherente al sistema de medición empleado, que depende del momento exacto en el que se miran las notas que suenan y varía de ejecución a ejecución. Analizar numerosas veces las obras probablemente estabilizaría los resultados.

6.2. Fuente de las partituras

Para las diferentes tandas de pruebas se creó una pequeña base de partituras, de la cual se extrajeron conjuntos de pruebas. No es difícil encontrar partituras libres de derechos de autor en internet; de hecho, existe un proyecto similar al proyecto Gutenberg: la Petrucci Music Library (Internet Music Score Library Project) [26], que recoge las primeras ediciones impresas de todas las obras que hayan pasado a dominio público. Por desgracia, la inmensa mayoría de las partituras no están disponibles en formato MuscXML, sino que se encuentran en PDF.

Aun así, existen otras páginas con cantidades considerables de partituras en MusicXML o formatos similares, particularmente en el ámbito coral (donde son comunes los arreglos y, por tanto, existe mayor demanda de partituras "editables"). Concretamente, las partituras para este proyecto se han extraído de las siguientes páginas:

Hausmusik.ch: Página de la asociación por el fomento de la *Hausmusik* (música interpretada en el hogar) en Alemania [27]. Posee una colección relativamente grande y ordenada de partituras para varios instrumentos, y generalmente en varios formatos como .cap (para el programa Capella), .mid, .pdf y .xml.

Choral Public Domain Library: La CPDL [28] es una biblioteca de partituras online en formato wiki especializada en obras corales. Fue creada en 1998, y a día de hoy está entre las colecciones de partituras libres más grandes del mundo, con más de 9000 obras individuales. Además suelen estar disponibles en varios formatos, como .mus (para el programa Finale), .sib (para el programa Sibelius), .mid, .pdf y .xml. Por desgracia, los formatos disponibles varían de obra a obra, por lo que encontrar una obra concreta en un formato concreto puede resultar difícil.

7. PLANIFICACIÓN Y PRESUPUESTO

Una vez explicada la totalidad del desarrollo del proyecto desde el inicio hasta las pruebas, se pasará a describir la planificación y el presupuesto que se empleó en éste.

7.1. Planificación

7.1.1. Planificación estimada

Al comenzar el proyecto se realizó una estimación general de los componentes en los que consistiría el mismo. La estructura general que se decidió seguir es idéntica a la que se ha empleado en esta memoria, y se muestra a continuación:

- Estudio inicial:
    - Estado del arte
    - Objetivos
- Análisis
    - Requisitos del sistema
    - Casos de uso
- Implementación
    - Implementación del parser
    - Elección del algoritmo de clasificación
    - Implementación de clasificador e interfaz
- Pruebas
- Documentación

Para el total del proyecto se estimó una duración de 105 días, con una interrupción de un mes durante el mes de junio debido a un viaje a Alemania para realizar pruebas de acceso a escuelas superiores de allí. A continuación, se muestra una tabla con la planificación inicial y el diagrama de Gantt.

Nombre	Fecha de inicio	Fecha de fin	Duración
Estudio inicial	2/03/15	19/03/15	14
Estado del arte	2/03/15	17/03/15	12
Definir objetivos	18/03/15	19/03/15	2
Análisis	20/03/15	30/03/15	7
Requisitos del sistema	20/03/15	25/03/15	4
Casos de uso	27/03/15	30/03/15	2
Implementación-Parte 1	31/03/15	29/05/15	44
Parser	31/03/15	20/05/15	37
Elección del algoritmo de clasificación	21/05/15	29/05/15	7
Parada en el desarrollo	1/06/15	30/06/15	22
Implementación-Parte 2	2/07/15	4/08/15	24
Interfaz y Clasificador	2/07/15	4/08/15	24
Pruebas	5/08/15	20/08/15	12
Documentación	2/03/15	25/08/15	127

Tabla 6: Planificación de tarea

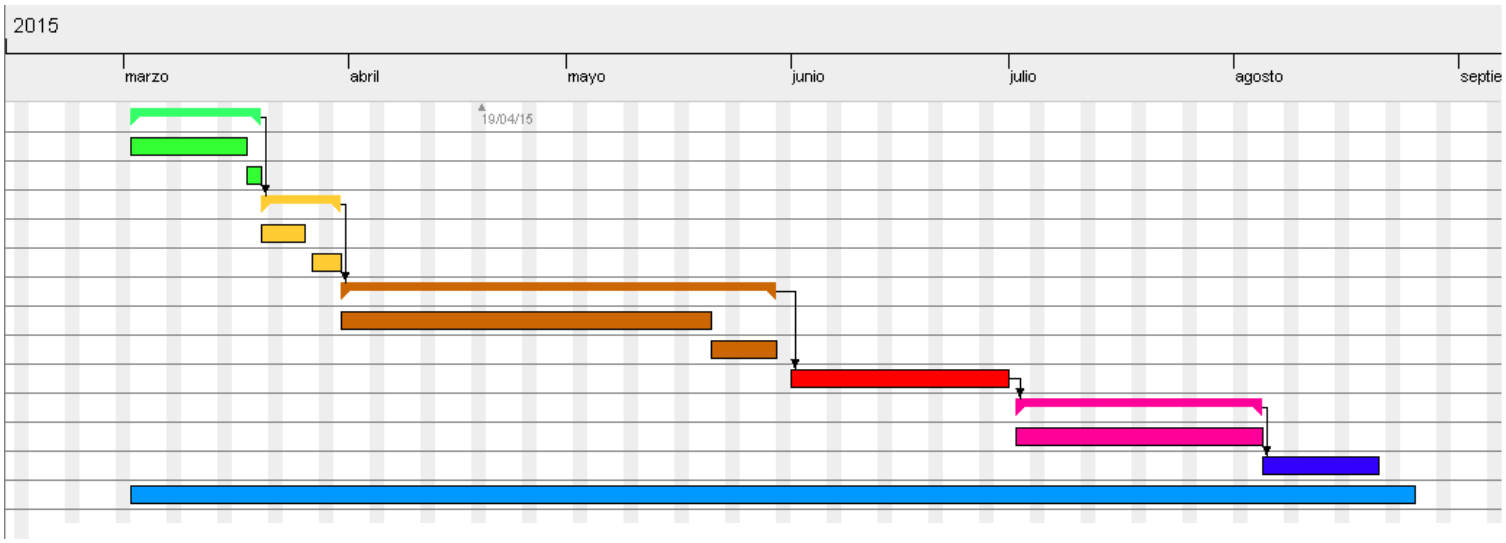


Tabla 7: Diagrama Gantt de la planificación

7.1.2. Duración real

Aunque se estimaron 105 días, al final el desarrollo llevó 113 debido a complicaciones en el desarrollo de la interfaz. A continuación se muestra la planificación y el diagrama finales.

Nombre	Fecha de inicio	Fecha de fin	Duración
Estudio inicial	2/03/15	19/03/15	14
Estado del arte	2/03/15	17/03/15	12
Definir objetivos	18/03/15	19/03/15	2
Análisis	20/03/15	30/03/15	7
Requisitos del sistema	20/03/15	25/03/15	4
Casos de uso	27/03/15	30/03/15	2
Implementación-Parte 1	31/03/15	29/05/15	44
Parser	31/03/15	20/05/15	37
Elección del algoritmo de clasificación	21/05/15	29/05/15	7
Parada en el desarrollo	1/06/15	30/06/15	22
Implementación-Parte 2	2/07/15	13/08/15	31
Interfaz y Clasificador	2/07/15	13/08/15	31
Pruebas	14/08/15	31/08/15	12
Documentación	1/07/15	2/09/15	46

Tabla 8: Duración real del proyecto

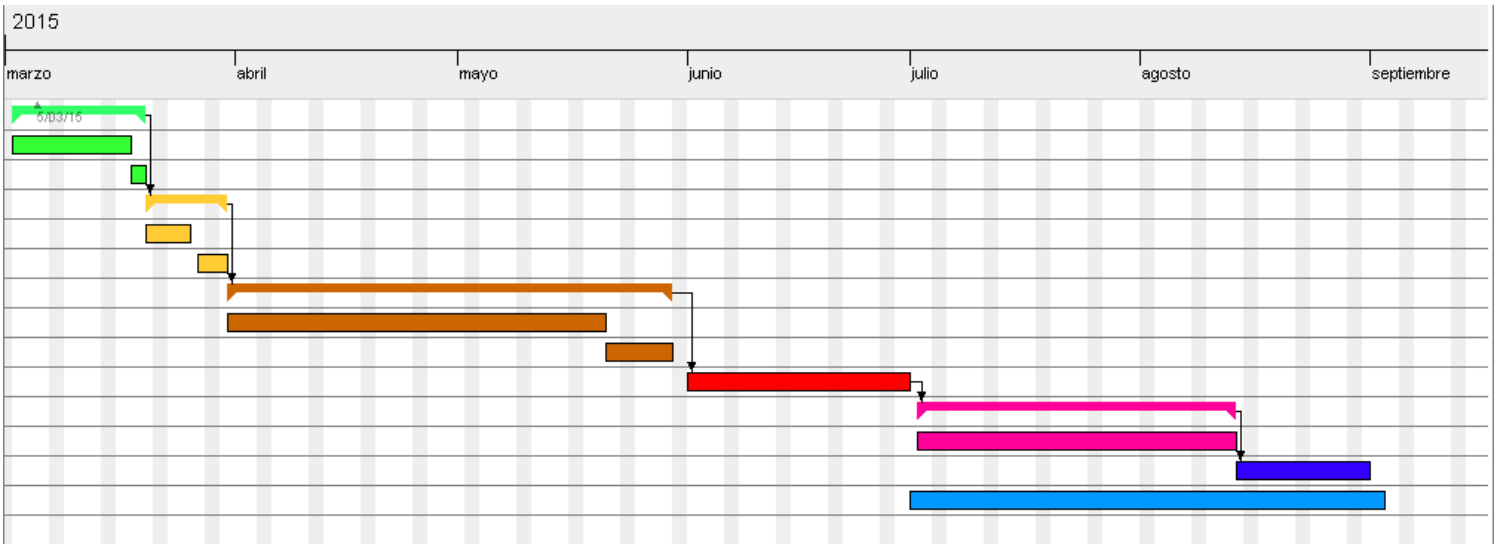


Tabla 9: Diagrama Gantt de la duración real

7.1.3. Metodología

Para este proyecto se decidió adoptar la metodología de desarrollo "Extreme Programming" (programación extrema) [29], una de las metodologías más importantes entre las metodologías de desarrollo de software ágil. Su flexibilidad y optimización hacia grupos de desarrollo pequeños la convirtió en la mejor candidata para este proyecto. Los valores que guían esta metodología son la simplicidad, la comunicación, la retroalimentación, el coraje y el respeto. Estos se traducen en diversas características del proceso desarrollo, que se explican a continuación, junto con cómo este proyecto se ha adherido a ellas:

Desarrollo iterativo e incremental. Se desarrolló el proyecto paso a paso, intentando pulir al máximo cada componente antes de pasar al siguiente.

Pruebas continuas. Pese a existir una fase de pruebas propiamente dicha al final del ciclo de desarrollo, a la hora de implementar el parser se realizaron regularmente pruebas para valorar el progreso de éste.

Programación en parejas. Por razones obvias esta recomendación no se ha tenido en cuenta al llevar a cabo este trabajo de fin de grado.

Integración del equipo de programación con el cliente. Se mantuvieron reuniones bisemanales con el equipo de tutores (en representación del cliente), además de mantenerse contacto por correo.

Refactorización del código. Durante el desarrollo del trabajo el código se refactorizó en numerosas ocasiones para corregir el funcionamiento o hacerlo más eficiente. Varios de los restos de estas refactorizaciones pueden encontrarse en el código fuente en forma de comentario al final de la clase para referencia.

Simplicidad en el código. Durante el desarrollo se buscó la mayor simplicidad posible en el código, procurando evitar objetos demasiado importantes y métodos excesivamente largos. De surgir estos, se dividían en la medida de lo posible en submétodos, buscando de una mayor legibilidad del código.

7.2. Presupuesto

En este apartado se discutirá el presupuesto del proyecto. Se ha enfocado desde una perspectiva unipersonal, donde el único componente del equipo asume varios roles.

7.2.1. Recursos humanos

Para este proyecto se repartirá la carga de trabajo en dos roles. Los salarios medios se han extraído de la página Jobtonic.es [30]:

Analista: Realizará el diseño del producto, especificará los requisitos y llevará a cabo la documentación. El salario medio de un analista es de 21.000€ anuales.

Programador: Se encargará de implementar las funcionalidades especificadas. El salario medio es de 18.000€ anuales.

Los sueldos especificados son anuales, divididos en 12 pagas quedarían en 1.750€/mes para el puesto de analista y 1.500€/mes para el de programador.

A continuación se muestra una división de las tareas y su respectivo equipo. Todos los costes se estiman partiendo de jornada completa (5 días semanales de 8 horas cada uno) salvo en el caso de la documentación: aquí solo los 5 últimos días se contabilizan como jornada completa y el resto se computa estimando 1h de trabajo por día.

Fase	Puesto	Días de trabajo	Coste
Estudio inicial	Analista	14	1.113,56€
Análisis	Analista	7	556,78€
Implementación 1	Programador	44	3.000€
Implementación 2	Programador	24	1.636,32€
Pruebas	Programador	12	818,16€
Documentación 1	Analista	101/8 = 12,6	1.002,27€
Documentación 2	Analista	5	397,70€
		TOTAL	8.522,79€

Tabla 10: Costes de recursos humanos

7.2.2. Recursos materiales

Para llevar a cabo el proyecto serán necesarios tanto equipo como software. El sistema no requiere una cantidad de recursos exagerada, aunque un procesador bueno y una cantidad suficiente de memoria RAM son deseables. En cuanto al software, sólo se emplearán las herramientas Eclipse y WEKA, ambas de uso gratuito.

Equipo	Precio(€) / unidad	Unidades	Período de amortización (meses)	Duración del proyecto (meses)	Coste final (€)
Lenovo Essential B50-80 (i7-500U / 8GB / 1TB / R5M230 / 15.6")	644€	1	24	6	161€
				TOTAL:	161€

Tabla 11: Costes de hardware

7.2.3. Costes totales

Para calcular el coste final del proyecto debe añadírsele el IVA, los beneficios y un porcentaje de riesgo para acomodar retrasos y costes no planeados.

Descripción	Coste
Recursos humanos	8.522,79€
Amortización del Hardware	161€
Amortización del software	0€
Total de costes directos	**8.683,79€**
Riesgo (20%)	1.736,76€
Beneficio(15%)	1.302,57€
Total sin IVA	**11.723,12€**
IVA (21%)	2.461,86€
Total con IVA	**14.184,98€**

Tabla 12: Costes totales

Por tanto, el coste final del proyecto es de:

14.184,98€

(Catorce mil ciento ochenta y cuatro euros con noventa y ocho céntimos)

Galapagar, a 1 de Marzo de 2015

Fdo. Pablo Carra García

Nota

La duración del proyecto resultó ser mayor de la prevista, pero el sobrecoste generado por ésto se pudo cubrir sin problemas con dinero para el riesgo calculado, evitando así una pérdida de beneficios.

8. MARCO REGULADOR Y ENTORNO SOCIO-ECONÓMICO

8.1. Marco regulador

En la actualidad no existe un marco legal concreto que afecte a la aplicación desarrollada. La ley de propiedad intelectual no se aplica, ya que no copia ni modifica los archivos que analiza, ni guarda partes de ellos. Tampoco existen servicios similares que hayan sido susceptibles de regulación.

En caso de plantearse un uso comercial, este probablemente sería en forma de *app* para Smartphone. Las aplicaciones están sujetas a la *Ley de Servicios de la Sociedad de la Información y del Comercio Electrónico* [31] general, y concretamente el *Dictamen 2/2013 del Grupo de Trabajo del artículo 29 de la Directiva 95/46* [32] de la Comisión Europea establece unas guías para el control de datos para dispositivos móviles. Son las siguientes:

Consentimiento previo a la instalación: El usuario deberá dar su consentimiento explícito antes de la introducción o extracción de datos del dispositivo, habiéndosele facilitado información clara y completa. Esta decisión deberá ser libre, es decir, debe ofrecérsele la opción de rechazar el tratamiento de sus datos junto a la opción de aceptarlo.

Limitación de la finalidad y minimización de datos: La finalidad del tratamiento de los datos deberá estar bien definida y ser comprensible por un usuario medio sin conocimientos jurídicos o técnicos especiales. También se exige a los desarrolladores que el tratamiento de datos personales solo pueda hacerse con una finalidad leal y lícita, que debe estar definida de antemano, y que solo se accederá a los datos estrictamente necesarios para la realización de la función deseada.

Seguridad: Los desarrolladores deberán tomar medidas durante el diseño de la aplicación para evitar el acceso no autorizado a datos personales, y garantizar tanto la protección de los datos en tránsito como de los almacenados. También se exigirá una evaluación continua de los riesgos actuales y futuros para la protección de datos, y la aplicación y evaluación de medidas correctoras eficaces.

Información: El usuario deberá poder saber la identidad legal del responsable del tratamiento de sus datos, además de qué tipo de datos personales están siendo tratados y con qué finalidad. También tendrá derecho a saber si esos datos se comunicarán a terceros, y en qué manera podrá ejercer sus derechos (retirar su consentimiento de uso y eliminar los datos).

Respeto a los derechos del interesado: El usuario deberá poder ejercer sus derechos de acceso, rectificación, supresión y bloqueo del tratamiento de datos. El responsable deberá informarle sobre qué datos se están tratando, la fuente de los mismos, y explicar la lógica detrás de cualquier tipo de decisiones automatizadas sobre los mismos que se puedan tomar. El acceso deberá ser fácil, y los usuarios deberán disponer de la posibilidad de retirar su consentimiento de una manera simple y cómoda.

Periodos de conservación: Los desarrolladores deberán considerar la conservación de los datos recogidos durante el uso de la aplicación y los riesgos que se plantean para su protección. También deberán tener en cuenta los datos de los usuarios que no hayan utilizado la aplicación en un tiempo prolongado, fijando un plazo a partir del cual la cuenta se considerará expirada. A partir de esa fecha, se deberá informar al usuario para ofrecerle recuperar sus datos personales o en su defecto anonimizarlos o suprimirlos de manera irreversible.

Uso por niños: Los desarrolladores deberán prestar atención al límite que define a los niños y a los menores de edad en las legislaciones nacionales, y en caso de que la aplicación esté destinada a ellos, prestar atención a las limitaciones de comprensión y atención de los menores al respecto del tratamiento de datos. Los responsables del tratamiento de estos datos deberán respetar de forma aún más rigurosa el principio de minimización de datos y limitación de la finalidad y, concretamente, no deberán procesar datos con fines de publicidad comportamental, ni directa ni indirectamente.

En su estado actual, la aplicación desarrollada no accedería ni recolectaría información de sus usuarios, pero si en un futuro esto se planteara de cara a una mejora del sistema, habría que respetar las directrices expuestas.

8.2. Entorno socio-económico

El principal objetivo de este proyecto es la investigación: ver hasta qué punto puede predecirse la música utilizando métodos estadísticos. Por tanto, el público objetivo de este trabajo serían otras personas que estuviesen investigando formas de predicción musical, aunque en principio también puede ser de utilidad a investigadores en otros campos, como la composición automática.

De desarrollarse una versión para uso público (cuya descripción detallada se encuentra en el apartado 9.2 “Vías de trabajo futuro”) el público objetivo serían músicos amateur interesados en improvisación grupal, principalmente de los géneros Jazz y Rock. Aunque es difícil estimar un número de músicos amateur o realizar un análisis demográfico conclusivo (aunque sería seguro asumir que el público objetivo tendría un sesgo joven), sí se pueden analizar los cambios que han surgido en esta industria en los últimos años.

Internet ha revolucionado el mundo de la música. Esta revolución va más allá de la crisis de la industria discográfica (un tema manido y no directamente relevante para este proyecto), sino que ha dado la vuelta al modelo productivo. La aparición de nuevos canales de venta de instrumentos a través de internet, que permiten bajar los precios gracias a ahorros en almacenaje y economías de escala, han facilitado la compra de instrumentos a un público mayor, sobre todo a principiantes. Páginas como thomann.com (la tienda de instrumentos online líder en Europa, con más de 6.700.000 clientes) venden instrumentos a precios que van desde los 75€ a los 10.000€.

Pero probablemente el cambio más grande haya sido la llegada de las redes sociales y la llamada web 2.0. Ha crecido un rico ecosistema de páginas que ofrecen a cualquier banda de música la posibilidad de hacer disponible su música a todo el que quiera escucharla, evitando cualquier tipo de mediación por agentes y discográficas. También han aparecido nuevas

formas de publicitarse, y de llegar directamente a nichos que los métodos tradicionales no permiten. La página que inició este cambio fue myspace.com, lanzada en 2003, que ofrecía a sus usuarios la posibilidad de mostrar un reproductor con canciones subidas por ellos. Pese a ser un página pensada para perfiles personales, muy pronto las bandas empezaron a usarla como plataforma en la que darse a conocer. Pese a que con los años la popularidad de MySpace ha decaído, han surgido numerosas páginas ya sí orientadas a subir música: bandcamp.com, reverbnation.com o soundcloud.com, una especie de youtube para pistas de audio. Pese a que estas páginas no han sustituido las vías "tradicionales" de llegar al éxito, sí han servido de lanzadera a ellas y han contribuido a crear una nutrida "segunda fila" de bandas de éxito local o nichos de sonido.

9. CONCLUSIONES Y PERSPECTIVAS DE DESARROLLO

En este apartado se repasarán los resultados obtenidos durante el desarrollo del proyecto. También se mencionarán posibles perspectivas de desarrollo futuro.

9.1. Revisión de los resultados conseguidos

El objetivo de este proyecto era crear un sistema capaz de predecir el progreso armónico de una obra a partir de los acordes que preceden a un momento concreto. Se decidió realizar la predicción a partir de un modelo estadístico, y para entrenarlo se emplearon obras corales de Beethoven, Haydn y Schubert. A continuación, se repasan los objetivos conseguidos:

Creación de un parser capaz de interpretar una partitura MusicXML: Se desarrolló satisfactoriamente un parser de XML, preparado para funcionamiento en tiempo real. Se prestó atención a la robustez, preparándolo para leer partituras de diferentes fuentes que proporcionaran menos información de la que especifica el estándar.

Extracción de información útil de la partitura: Se empleó conocimiento experto para decidir qué información de la partitura se emplearía para la predicción. El parser fue diseñado de tal manera que un cambio de criterio fuera fácilmente implementable. Esto quedó demostrado al añadirse el control del bajo de cada acorde, información no planeada en el primer diseño y cuya implementación no retrasó el proyecto.

Categorización los acordes: Se desarrolló un sistema experto que asigna el acorde correspondiente a las notas extraídas de la partitura en cada momento. El sistema es capaz de reconocer 8 tipos de acordes distintos, pero su sistema basado en plantillas permite añadir más con facilidad.

Creación una base de datos para generar el modelo: Se creó una base de datos con un total de 1568 entradas a partir de 20 obras distintas. También se crearon bases de datos de otras épocas como referencia, aunque no se emplearan para crear el modelo de predicción.

Selección de un algoritmo de predicción: Se evaluaron un total de 9 algoritmos distintos a la hora de decidir cuál emplear para el predictor. El algoritmo elegido, el C4.5, alcanzó una mejora del 15% sobre el clasificador base en el momento de predecir una de entre las 24 clases distintas.

Interfaz de usuario: Se desarrolló una interfaz de usuario gráfica para el manejo del programa, que ofrece información y permite manejo en tiempo real. Se diseñó de tal manera que pudiera ser empleada por un usuario sin conocimientos técnicos.

En conclusión: Tal y como se expuso en los objetivos, **se ha desarrollado un predictor armónico completamente funcional.** Pese a que los resultados no permiten aún un uso práctico fuera del ámbito de la investigación, son sin duda prometedores, y ofrecen una base sobre la que seguir construyendo. También se ha demostrado que predicciones realizadas sobre información extraída de una ventana de tiempo discreta pueden ser correctas. El

sistema no necesita información sobre la estructura de la obra, ni precisa conocer su desarrollo completo.

Desde esta posición se ofrecen caminos de desarrollo futuro, que se detallarán en el apartado siguiente.

9.2. Vías de trabajo futuro

La aplicación desarrollada ofrece numerosas posibilidades de evolución. Estas se pueden dividir en dos grupos: mejorar la predicción o convertir el programa en una aplicación de uso general. Por supuesto, ambas posibilidades están entrelazadas: no se podría plantear una aplicación de uso general antes de aumentar el porcentaje de aciertos en las predicciones, y por otro lado, el enfoque eminentemente práctico que se ha dado al predictor limita su interés puramente científico.

9.2.1. Perfeccionamiento del predictor

Pese a ofrecer ya a día de hoy resultados dignos, sería capital ver hasta qué punto puede mejorarse el porcentaje de aciertos del predictor. Sería deseable superar la barrera del 50%, aunque para plantearse un uso generalizado habría que llegar al menos al 70%. Ésto se podría enfocar desde varias perspectivas.

Replantear la información que se extrae de la partitura: Podrían buscarse otros indicadores del desarrollo de la obra, modificar el número de compases que se analizan antes de realizar una predicción, o añadir el reconocimiento de acordes menos comunes. A no ser que se encontrara un atributo nuevo de muy alta significancia que no se haya tenido en cuenta hasta ahora, este es el enfoque que a primera vista resulta menos prometedor: el aumento de los datos ralentizaría el funcionamiento y un exceso de atributos podría facilitar el sobreaprendizaje, cuando la idea del proyecto siempre ha sido tender a la abstracción y la generalización.

Modificar la manera en la que se decide qué acorde es el que está sonando: Se podría pasar del método actual, que toma una "fotografía" del instante, a uno que recopilara todas las notas que suenen durante un intervalo de tiempo, pondere su importancia en base a valores como su duración o cuantas veces se repite una nota, y a partir de ahí intentara inferir de qué acorde se trata. Cuánto podría mejorar esto la predicción general es difícil de estimar *a priori*: al fin y al cabo, el error del método "fotográfico" para determinar el acorde es muy difícil de evaluar, ya que no es posible saber compás por compás si se ha tomado la medida precisamente en un instante que no es representativo. Además, el sistema estaría expuesto a otros riesgos: realizar la medición en una ventana de tiempo abre la posibilidad a que en una misma medición se entremezclen dos acordes consecutivos, llevando a una clasificación errónea. No se podría establecer una predicción en base al compás entero, ya que frecuentemente se cambia de acorde varias veces en el mismo compás, y por tanto la duración del fragmento a analizar sería un dilema en sí mismo: si es demasiado largo, se corre el riesgo recién expuesto, si es demasiado corto, pierde su sentido al parecerse cada vez más al método "fotográfico".

Implementar un detector de modulaciones: Aunque la modulación al relativo menor está representada en el predictor, si la obra modula a una tonalidad distinta el

predictor la analiza como si siguiera en la original, lo cual genera ruido en los datos, ya que las progresiones dejan de tener sentido. Un detector de modulaciones que pudiera identificar una modulación y "traducir" los grados a la tonalidad nueva no sólo acabaría con ese ruido, sino que lo transformaría en datos útiles. Por desgracia, un sistema que no sólo detecte una modulación en el tiempo en el que ocurre, sino que además pueda interpretar correctamente a cuál de todas las tonalidades existentes ha modulado es algo muy difícil de desarrollar en la práctica. Incluso cuando se realizan análisis manuales de obras, las modulaciones suelen determinarse *a posteriori*: Se busca el punto en el que queda definitivamente claro que se está en una nueva tonalidad y cuál es (generalmente, buscando una cadencia V-I), y a partir de ahí se recorre la partitura hacia atrás en el tiempo, asignando los grados que corresponden. Hacerlo en tiempo real sería un reto que bien supondría un proyecto en sí mismo.

9.2.2. Conversión en aplicación de uso general

Una vez aumentado el porcentaje de predicciones correctas, se plantearía el uso para el público general. Especialmente útil podría ser para músicos de los campos del Jazz y del Rock. En estos ámbitos, las "jams" (improvisaciones grupales) son comunes, y la aplicación desarrollada, si se mantuviera ejecutando durante la interpretación, podría servir de *chivato* para un músico desorientado en una jam estructurada (donde la progresión está fijada de antemano) u ofrecer sugerencias en jam libres (donde no hay una progresión fija). Por supuesto, para llevar esto a cabo habría que resolver los siguientes problemas:

Entrada de audio real: Sería necesario desarrollar un sistema capaz de trabajar con una entrada de audio a través de micrófono. Aunque un afinador (es decir, un micrófono con un programa que determina la nota que suena y si está afinada) es una aplicación informática sencilla y un estándar en la industria musical desde hace años, identificar varias notas simultáneas aumenta la dificultad exponencialmente. Al describir el estado del arte se mencionaron dos proyectos que intentan precisamente eso, y con buenos resultados. Otro posible enfoque sería emplear el software S.P.E.A.R [33]: al dividir el sonido en los armónicos que lo componen, podría intentarse deducir de los que con más fuerza suenan (y, por tanto, son los que más notas comparten) qué acorde está sonando, sin llegar a necesitar analizar una por una todas las notas que podrían sonar a la vez. El sistema se haría una "impresión general" del ambiente sonoro y trabajaría con ella. Por supuesto, se eligiera el método que se eligiera, habría que ver si estos análisis se podrían realizar en tiempo real.

Adaptación a Smartphones: Si se consiguiera desarrollar una entrada de audio real, el siguiente reto sería realizar un *port* a smartphones, ya que el público objetivo no tendría que tener necesariamente grandes conocimientos técnicos, y usaría la aplicación como un accesorio, no como un fin en sí mismo. En este caso, resultaría un ventaja haber desarrollado la aplicación Java, ya que su adaptación a otros dispositivos sería mucho más sencilla. Quedaría por ver si sería posible ejecutar la aplicación en un dispositivo de menor potencia sin perder su funcionamiento en tiempo real, indispensable para su utilidad.

Evaluación de su funcionamiento con otros géneros y épocas: Se podría evaluar el funcionamiento del sistema con otros géneros de música, y no sólo los ya mencionados Jazz y Rock: de encontrar la manera de reconocer las modulaciones se podrían analizar, por ejemplo, obras del romanticismo con ciertas esperanzas de éxito.

En caso de no funcionar con la exactitud deseada, se podría probar entrenando el modelo con una base de datos más reciente. Esto plantearía nuevos problemas, ya que la inmensa mayoría de la música más moderna aún no está libre de derechos de autor, y crear un conjunto de entrenamiento útil podría resultar caro. Además, este tipo de música es más difícil de encontrar en partitura: suelen comercializarse en cifrado o en tablatura, las cuales requerirían un parser distinto.

Desarrollo de un acompañante automático: De llevarse a cabo con éxito las propuestas anteriores, podría afrontarse un nuevo reto: crear un sistema que no sólo predijese el acorde que va a venir, sino que lo tocase, colaborando activamente en la interpretación. En un principio este acompañamiento sería en forma de acordes en bloque, pero en un futuro, y después de experimentación, se podrían afrontar acompañamientos más ricos. Cabe destacar que con nuestro sistema solo se podría generar un acompañamiento a partir de un conjunto multi-instrumental, o que al menos proporcionara un tejido armónico. Crear un acompañamiento en tiempo real partir de una melodía es una línea de investigación completamente diferente.

Como se ha podido ver, el sistema implementado ofrece numerosas vías de desarrollo, y pone la primera piedra en un camino cuyo recorrido puede ofrecer resultados apasionantes.

10. REFERENCIAS

[1] E. Rueda, "Los enlaces de la escala. Enlace de acordes", en *Armonía,* 1ª Edición, Madrid, España; Ed. Real Musical, 1998, pp. 12-14.

[2] J. Cabero Piris, "Funciones Tonales. Composición" en *Visualización, armonía e improvisación,* 1ª Edición, Madrid, España; Ed. Javier Cabero Piris Igualada, 2011, pp. 189- 193.

[3] D. Cope, *Experiments in Musical Intelligence*, Madison, Wisconsin; A–R Editions, 1996, p.7.

[4] E.Bowles*, Musicke's Handmaiden: Or Technology in the Service of the Arts* , Ithaca, New York; Cornell University Press, 1970.

[5] A. Alpern, *Techniques for Algorithmic Composition of Music*, Springfield, Massachusets; Hampshire College Divisional Examination, Humanities and Arts, 1995.

[6] I. Xenakis, *Formalized Music: Thought and Mathematics in Composition*, *Harmonologia series nº 6*; Hillsdale, NY; Ed. Pendragon Press, 2001.

[7] P.M. Gibson, J.A. Byrne, "NEUROGEN, Musical Composition Using Genetic Algorithms and Cooperating Neural Networks", presentado en la *Second International Conference on Neural Networks,* Bournemouth, UK; 18-20 Nov, 1991.

[8] C. Sánchez Quintana, F. Moreno Arcas, D. Albarracín Molina, J.D. Fernández Rodríguez, F. J. Vico, " Melomics: A Case Study of AI in Spain" en *AI Magazine*, Vol. 34 No. 3, Palo Alto, California; 2013, pp. 99 – 103.

[9] B. A. Pardo, W. Birmingham, "Following a Musical Performance from a Partially Specified Score", en *Proceedings of the 2001 Multimedia Technology Applications Conference*, Irvine, California; 2001.

[10] B. A. Pardo, W. Birmingham, "Modeling Form for On-Line Following of Musical Performances", en *AAAI'05 Proceedings of the 20th national conference on Artificial Intelligence*, Pittsburgh, Pennsylvania, 09- 13 de Julio, 2005, pp. 1018- 1023

[11] H. Papadopoulos, G. Peeters,, "Chord Estimations Using Chord Templates and HMM" ,presentado en *MIREX*, Philadelphia, Pennsylvania; 14-18 Sept, 2008.

[12] Yushi Ueda, Yuki Uchiyama, Takuya Nishimoto, Nobutaka Ono, Shigeki Sagayama, "HMM-Based Approach for Automatic Chord Detection Using Refined Acoustic Features", presentado en *IEEE International Conference on Acoustics Speech and Signal Processing(ICASP)*, Dallas, Texas; 14- 19 Mar, 2010.

[13] W. Bas De Haas**,** José Pedro Magalhães, Frans Wiering, Remco C. Veltkamp, "Automatic Functional Harmonic Analysis", en *Computer Music Journal*, Vol. 37 No. 4 Cambridge, Massachusets; 2013, pp. 37-53.

[14] M.L.Johnson, "Towards an Expert System for Expressive Performance" en *Computer,* Vol. 24 No.7, Stanford University, California; 1991, pp. 30-34

[15] J.L. Arcos, R. López de Mántaras, X. Serra, "Generating Expressive Musical Performances with SaxEx", en *Journal of New Music Research,* Vol. 27 No.3, Taylor & Francis Online Publishing, 1998, pp. 194-210

[16] A. Friberg, V. Colombo. L. Frydén, J. Sundberg, "Generating Musical Performances with Director Musices", en *Computer Music Journal*, Vol.24 No. 3, Cambridge, Massachusets, 2000, pp. 23-29

[17] A. Benetti, "Expressivity and Musical Performance: Practice Strategies for Pianists", presentado en la *Performance Studies Network International Conference*, Cambridge , 4-7 Abril, 2013

[18] Protocolo MIDI. Disponible [online] http://www.midi.org/

[19] Formato MusicXML. Disponible [online] http://www.musicxml.com/

[20] WEKA Data Mining Software. Disponible [online] http://www.cs.waikato.ac.nz/ml/weka/

[21] JAVA. Disponible [online] https://www.oracle.com/java/index.html

[22] C. Shearer, "The CRISP-DM Model: the New Blueprint for Data Mining", en *International Journal of Data Warehousing*, Monash University, Australia; 2000, pp. 13-22

[23] WEKA Classification Algorithms Plug-In. Disponible [online] http://wekaclassalgos.sourceforge.net/

[24] R. Kohavi, "The Power of Decision Tables" en *Proceedings European Conference on Machine Learning, Lecture Notes in Artificial Intelligence*, Vol. 914, Ed. Springer, Berlín; 1995, pp. 174-189.

[25] H. Shi, "Best-First Decision Tree Learning" Tesis de Máster, University of Waikato, Hamilton, Nueva Zelanda; 2007.

[26] Petrucci Music Library. Disponible [online] http://imslp.org/

[27] Verein zur Förderung der Hausmusik. Disponible [online] www.hausmusik.ch

[28] Choral Public Domain Library. Disponible [online] http://www3.cpdl.org

[29] K. Beck, *Extreme Programming explained: embrace change*, Addison-Wesley Longman Publishing Co. Inc., Boston, Massachusetts, 2000

[30] Jobtonic. Disponible [online] http://www.jobtonic.es/

[31] Ley de Servicios de la Sociedad de la Información y del Comercio Electrónico. Disponible [online] http://www.lssi.gob.es/paginas/Index.aspx

[32] Dictamen 02/2013 sobre las aplicaciones de los dispositivos inteligentes. Disponible [online] http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2013/wp202_es.pdf

[33] M. K. Klingbeil, "Spectral Analysis, Editing and Resynthesis: Methods and Applications" Disertación doctoral, Columbia University, New York; 2009

ANEXO I: REQUISITOS

AI.1. Requisitos funcionales

ID: RF-01	Tipo: Funcional	Necesidad: Alta
Título: Menú		
Prioridad: Alta	**Fuente**: Tutor	
Claridad: Alta	**Verificabilidad**: Alta	
Estabilidad: Durante toda la vida del software.		
Descripción: La aplicación debe tener un menú en forma de interfaz gráfica		

ID: RF-02	Tipo: Funcional	Necesidad: Alta
Título: Selector de archivos		
Prioridad: Alta	**Fuente**: Tutor	
Claridad: Alta	**Verificabilidad**: Alta	
Estabilidad: Durante toda la vida del software.		
Descripción: La aplicación debe permitir elegir un archivo a través de una ventana de buscador de archivos		

ID: RF-03	Tipo: Funcional	Necesidad: Alta
Título: Play		
Prioridad: Alta	**Fuente**: Tutor	
Claridad: Alta	**Verificabilidad**: Alta	
Estabilidad: Durante toda la vida del software.		
Descripción: La aplicación debe tener un botón que permita iniciar la reproducción del archivo		

ID: RF-04	Tipo: Funcional	Necesidad: Alta
Título: Stop		
Prioridad: Alta	**Fuente**: Tutor	
Claridad: Alta	**Verificabilidad**: Alta	
Estabilidad: Durante toda la vida del software.		

Descripción: La aplicación debe tener un botón que permita detener la reproducción en cualquier momento

ID: RF-05	**Tipo: Funcional**	**Necesidad: Alta**
Título: Estadísticas		
Prioridad: Alta	**Fuente**: Tutor	
Claridad: Alta	**Verificabilidad**: Alta	
Estabilidad: Durante toda la vida del software.		
Descripción: La aplicación debe mostrar las estadísticas de acierto/error de la ejecución actual		

ID: RF-06	**Tipo: Funcional**	**Necesidad: Alta**
Título: Feedback Visual		
Prioridad: Alta	**Fuente**: Tutor	
Claridad: Alta	**Verificabilidad**: Alta	
Estabilidad: Durante toda la vida del software.		
Descripción: La aplicación debe representar el estado actual de la reproducción y las previsiones de una manera clara e intuitiva		

AI.2. Requisitos no funcionales

ID: RNF-01	**Tipo: No Funcional**	**Necesidad: Alta**
Título: Portabilidad		
Prioridad: Alta	**Fuente**: Tutor	
Claridad: Alta	**Verificabilidad**: Alta	
Estabilidad: Durante toda la vida del software.		
Descripción: La aplicación debe ser portable a distintos dispositivos		

ID: RNF-02	**Tipo: No Funcional**	**Necesidad: Alta**
Título: Usabilidad		
Prioridad: Alta	**Fuente**: Tutor	
Claridad: Alta	**Verificabilidad**: Alta	

Estabilidad: Durante toda la vida del software.
Descripción: La aplicación debe ser usable por un usuario sin grandes conocimientos técnicos.

ID: RNF-03	**Tipo: No Funcional**	**Necesidad: Alta**
Título: Fichero ejecutable		
Prioridad: Alta	**Fuente**: Tutor	
Claridad: Alta	**Verificabilidad**: Alta	
Estabilidad: Durante toda la vida del software.		
Descripción: La aplicación deberá poder iniciarse desde un fichero ejecutable		

ANEXO II: CASOS DE USO:

Identificador	**CU-01**
Título	Cargar Archivo
Descripción	El usuario decide sobre qué archivo ejecutar la predicción
Actor	Usuario
Precondiciones	Ninguna
Postcondiciones	Se sabe que archivo se va a analizar
Escenario	El usuario inicia la aplicación y selecciona el archivo con el que quiere trabajar.

Identificador	**CU-02**
Título	Comenzar análisis
Descripción	El usuario inicia el análisis
Actor	Usuario
Precondiciones	Cargar archivo
Postcondiciones	Se ha realizado el análisis y la predicción
Escenario	El usuario inicia la aplicación, elige un archivo y comienza el análisis pulsando el botón “Play”. Se reproduce el archivo y se ejecutan los análisis.

Identificador	**CU-03**
Título	Detener análisis
Descripción	El usuario detiene el análisis
Actor	Usuario
Precondiciones	Comenzar análisis

Postcondiciones	Se han detenido reproducción y análisis
Escenario	El usuario inicia la aplicación, elige un archivo y comienza el análisis. Antes de que éste haya terminado, decide detenerlo y presiona el botón “Stop” para ello.

Identificador	**CU-04**
Título	Salir
Descripción	El usuario sale de la aplicación
Actor	Usuario
Precondiciones	Ninguna
Postcondiciones	Se ha cerrado la aplicación
Escenario	El usuario inicia la aplicación y sin necesariamente haber realizado antes ningún análisis elige la opción “Salir”. La aplicación se cierra.

ANEXO III: RESULTADOS COMPLETOS DE LOS CLASIFICADORES

AIII.1. Clasificador ZeroR:

Correctly Classified Instances	347	22.1301 %
Incorrectly Classified Instances	1221	77.8699 %
Kappa statistic	0	
Mean absolute error	0.0741	
Root mean squared error	0.1923	
Relative absolute error	100%	
Root relative squared error	100%	
Total Number of Instances	1568	

Precisión detallada por clase

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	1	1	0.221	1	0.362	0.495	IM
	0	0	0	0	0	0.25	Im
	0	0	0	0	0	0.493	IIM
	0	0	0	0	0	0.499	IIm
	0	0	0	0	0	0.454	bIIM
	0	0	0	0	0	0.47	bIIm
	0	0	0	0	0	0.488	IIIM
	0	0	0	0	0	0.46	IIIm
	0	0	0	0	0	0.499	bIIIM
	0	0	0	0	0	0.437	bIIIm
	0	0	0	0	0	0.492	IVM
	0	0	0	0	0	0.449	IVm
	0	0	0	0	0	0.495	VM
	0	0	0	0	0	0.458	Vm
	0	0	0	0	0	0.485	bVM
	0	0	0	0	0	0.433	bVm
	0	0	0	0	0	0.493	VIM
	0	0	0	0	0	0.475	VIm
	0	0	0	0	0	0.46	bVIM
	0	0	0	0	0	0.425	bVIm
	0	0	0	0	0	0.49	VIIM
	0	0	0	0	0	0.499	VIIm
	0	0	0	0	0	0.477	bVIIM
	0	0	0	0	0	0.449	bVIIm
Weighted Avg.	0.221	0.221	0.049	0.221	0.08	0.487	

Ilustración 18: Precisión detallada por clase del clasificador ZeroR

Matriz de Confusión

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	<-- classified as
347	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	a = IM
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	b = Im
69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	c = IIM
60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	d = IIm
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	e = bIIM
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	f = bIIm
72	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	g = IIIM
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	h = IIIm
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	i = bIIIM
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	j = bIIIm
137	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	k = IVM
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	l = IVm
305	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	m = VM
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	n = Vm
73	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	o = bVM
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	p = bVm
79	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	q = VIM
43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	r = VIm
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	s = bVIM
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	t = bVIm
88	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	u = VIIM
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	v = VIIm
56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	w = bVIIM
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x = bVIIm

Ilustración 19: Matriz de confusión del clasificador ZeroR

AIII.2. Clasificador por tabla de decisión:

Correctly Classified Instances	512	32.6531 %
Incorrectly Classified Instances	1056	67.3469 %
Kappa statistic	0.209	
Mean absolute error	0.0714	
Root mean squared error	0.1866	
Relative absolute error	96.3266 %	
Root relative squared error	97.0359 %	
Total Number of Instances	1568	

Precisión detallada por clase

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.611	0.305	0.363	0.611	0.455	0.728	IM
	0	0.001	0	0	0	0.451	Im
	0.203	0.011	0.452	0.203	0.28	0.584	IIM
	0.067	0.016	0.143	0.067	0.091	0.727	IIm
	0	0.008	0	0	0	0.808	bIIM
	0.071	0.011	0.105	0.071	0.085	0.73	bIIm
	0.139	0.024	0.217	0.139	0.169	0.66	IIIM
	0	0.005	0	0	0	0.551	IIIm
	0.35	0.019	0.326	0.35	0.337	0.773	bIIIM
	0.118	0.005	0.2	0.118	0.148	0.754	bIIIm
	0.314	0.079	0.276	0.314	0.294	0.724	IVM
	0	0.001	0	0	0	0.538	IVm
	0.446	0.182	0.372	0.446	0.405	0.697	VM
	0	0	0	0	0	0.616	Vm
	0.192	0.025	0.269	0.192	0.224	0.814	bVM
	0	0	0	0	0	0.641	bVm
	0.063	0.027	0.111	0.063	0.081	0.632	VIM
	0	0.001	0	0	0	0.586	VIm
	0.037	0.003	0.2	0.037	0.063	0.708	bVIM
	0	0.002	0	0	0	0.726	bVIm
	0.432	0.049	0.342	0.432	0.382	0.826	VIIM
	0	0.001	0	0	0	0.637	VIIm
	0.304	0.017	0.405	0.304	0.347	0.833	bVIIM
	0	0.001	0	0	0	0.778	bVIIm
Weighted Avg.	0.327	0.119	0.28	0.327	0.29	0.711	

Ilustración 20: Precisión detallada de la tabla de decisión

Matriz de Confusión

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	<-- classified as
212	1	1	6	1	0	2	1	1	0	25	0	85	0	0	0	3	0	0	0	1	0	8	0	a = IM
3	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	b = Im
18	0	14	2	0	0	1	0	1	0	10	0	16	0	1	0	3	0	0	0	1	0	2	0	c = IIM
27	0	1	4	0	1	3	0	0	0	12	0	4	0	1	0	3	0	0	0	2	0	2	0	d = IIm
4	0	0	0	0	0	0	0	0	1	1	1	1	0	3	0	4	0	0	1	2	0	0	0	e = bIIM
6	0	0	1	2	2	0	1	1	1	1	0	3	0	0	0	1	0	0	0	9	0	0	0	f = bIIm
21	0	1	3	0	2	10	0	4	0	2	0	11	0	1	0	5	1	0	0	8	2	1	0	g = IIIM
9	0	1	1	0	0	0	0	2	0	2	0	10	0	0	0	0	0	0	1	1	0	0	0	h = IIIm
8	0	0	0	0	0	0	1	14	0	1	0	5	0	4	0	0	0	2	0	5	0	0	0	i = bIIIM
7	0	0	1	1	0	0	0	0	2	0	0	2	0	1	0	0	0	0	1	2	0	0	0	j = bIIIm
56	0	1	1	0	0	5	0	1	0	43	0	22	0	1	0	4	1	0	0	0	0	2	0	k = IVM
2	0	0	0	0	1	1	0	0	0	0	0	2	0	1	0	0	0	0	0	1	0	1	0	l = IVm
106	0	6	2	0	1	2	2	2	1	24	0	136	0	5	0	13	0	0	0	5	0	0	0	m = VM
7	0	1	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	n = Vm
10	0	2	1	3	4	5	0	3	2	0	0	1	0	14	0	2	0	0	0	24	0	2	0	o = bVM
8	0	0	0	1	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	p = bVm
19	0	0	3	0	2	5	1	0	1	7	0	24	0	3	0	5	0	0	0	5	0	3	1	q = VIM
21	0	2	0	0	0	3	0	0	0	5	0	9	0	0	0	1	0	1	0	0	0	1	0	r = VIm
4	0	0	0	0	2	1	0	3	0	2	0	7	0	1	0	0	0	1	0	3	0	3	0	s = bVIM
3	0	0	0	1	1	1	0	3	1	0	0	2	0	1	0	0	0	0	0	3	0	0	0	t = bVIm
9	0	0	2	3	3	3	0	6	1	2	0	5	0	15	0	0	0	1	0	38	0	0	0	u = VIIM
5	0	1	0	0	0	3	0	0	0	0	0	10	0	0	0	1	0	0	0	0	0	0	0	v = VIIm
16	0	0	1	0	0	0	0	1	0	16	0	5	0	0	0	0	0	0	0	0	0	17	0	w = bVIIM
3	0	0	0	0	0	1	2	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	x = bVIIm

Ilustración 21: Matriz de confusión de la tabla de decisión

AIII.3. Clasificador BFTree:

Correctly Classified Instances	525	33.4821 %
Incorrectly Classified Instances	1043	66.5179 %
Kappa statistic	0.2278	
Mean absolute error	0.0598	
Root mean squared error	0.2065	
Relative absolute error	80.7606 %	
Root relative squared error	107.3684 %	
Total Number of Instances	1568	

Precisión detallada por clase

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.599	0.233	0.423	0.599	0.496	0.727	IM
	0	0.002	0	0	0	0.537	Im
	0.145	0.046	0.127	0.145	0.135	0.612	IIM
	0.067	0.02	0.118	0.067	0.085	0.631	IIm
	0	0.015	0	0	0	0.489	bIIM
	0.071	0.012	0.095	0.071	0.082	0.59	bIIm
	0.167	0.041	0.162	0.167	0.164	0.603	IIIM
	0.037	0.013	0.048	0.037	0.042	0.501	IIIm
	0.225	0.012	0.333	0.225	0.269	0.583	bIIIM
	0.176	0.007	0.214	0.176	0.194	0.668	bIIIm
	0.255	0.051	0.324	0.255	0.286	0.695	IVM
	0	0.001	0	0	0	0.49	IVm
	0.498	0.196	0.381	0.498	0.432	0.689	VM
	0.091	0.002	0.25	0.091	0.133	0.663	Vm
	0.233	0.027	0.298	0.233	0.262	0.637	bVM
	0.25	0.002	0.5	0.25	0.333	0.745	bVm
	0.139	0.021	0.256	0.139	0.18	0.658	VIM
	0.047	0.01	0.111	0.047	0.066	0.549	VIm
	0	0.005	0	0	0	0.576	bVIM
	0.25	0.008	0.25	0.25	0.25	0.713	bVIm
	0.398	0.03	0.438	0.398	0.417	0.699	VIIM
	0.1	0.003	0.286	0.1	0.148	0.644	VIIm
	0.25	0.012	0.438	0.25	0.318	0.708	bVIIM
	0	0.001	0	0	0	0.569	bVIIm
Weighted Avg.	0.335	0.105	0.309	0.335	0.312	0.667	

Ilustración 22: Precisión detallada del BFTree

Matriz de Confusión

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	<-- classified as
208	1	14	9	0	1	4	6	0	1	17	1	67	0	1	0	6	5	0	1	0	1	4	0	a = IM
4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	b = Im
16	0	10	2	1	1	3	1	1	0	4	0	24	0	0	0	1	2	1	0	1	0	1	0	c = IIM
21	0	4	4	1	0	1	4	0	0	8	0	10	0	0	0	3	1	1	0	0	0	2	0	d = IIm
0	1	0	0	0	2	2	0	0	0	0	0	3	0	2	1	2	0	0	0	5	0	0	0	e = bIIM
4	0	2	1	2	2	4	0	0	2	0	0	6	0	2	0	1	0	0	0	1	0	1	0	f = bIIm
13	0	4	3	1	2	12	1	1	0	4	0	14	0	7	0	1	2	0	0	6	0	1	0	g = IIIM
10	0	1	0	1	1	2	1	1	0	1	0	8	0	0	0	1	0	0	0	0	0	0	0	h = IIIm
3	0	4	2	1	3	4	0	9	1	1	0	2	0	2	0	0	0	0	3	5	0	0	0	i = bIIIM
1	0	1	0	1	0	4	0	0	3	0	0	2	0	0	0	0	0	0	1	4	0	0	0	j = bIIIm
52	0	6	3	0	1	7	1	0	0	35	0	26	1	0	0	2	1	0	1	0	0	1	0	k = IVM
2	0	0	0	1	0	2	0	0	0	1	0	2	0	0	0	0	0	0	0	0	0	1	0	l = IVm
86	0	11	4	1	0	8	3	2	1	19	0	152	1	1	0	8	4	0	0	1	3	0	0	m = VM
6	0	2	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	n = Vm
6	0	1	1	2	4	6	0	4	1	3	0	8	0	17	1	2	0	1	1	15	0	0	0	o = bVM
1	0	0	0	0	0	2	1	0	0	0	0	4	0	1	3	0	0	0	0	0	0	0	0	p = bVm
16	0	6	1	6	0	4	0	0	0	4	0	21	1	2	0	11	1	0	1	2	0	2	1	q = VIM
13	0	4	0	1	0	2	1	0	0	3	0	12	0	0	1	0	2	1	0	0	0	3	0	r = VIm
4	0	0	1	1	2	1	0	2	1	1	0	8	0	2	0	0	0	0	0	3	0	1	0	s = bVIM
1	0	1	1	0	0	1	0	2	2	0	1	2	0	0	0	0	0	0	4	1	0	0	0	t = bVIm
3	0	4	1	2	1	4	0	3	0	3	0	6	0	18	0	2	0	3	2	35	0	1	0	u = VIIM
6	1	2	0	0	0	0	0	0	0	1	0	8	0	0	0	0	0	0	0	0	2	0	0	v = VIIm
15	0	2	0	0	1	1	1	2	1	2	0	12	0	0	0	3	0	0	0	1	1	14	0	w = bVIIM
1	0	0	0	1	0	0	1	0	1	0	0	1	0	2	0	0	0	0	2	0	0	0	0	x = bVIIm

Ilustración 23: Matriz de confusión del BFTree

AIII.4. Clasificador ID3:

Correctly Classified Instances	551	35.1403 %
Incorrectly Classified Instances	550	35.0765 %
Kappa statistic	0.425	
Mean absolute error	0.0415	
Root mean squared error	0.202	
Relative absolute error	80.4828 %	
Root relative squared error	126.4777 %	
UnClassified Instances	467	29.7832 %
Total Number of Instances	1568	

Precisión detallada por clase

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.619	0.132	0.601	0.619	0.61	0.699	IM
	0	0.001	0	0	0	0.5	Im
	0.356	0.027	0.364	0.356	0.36	0.606	IIM
	0.441	0.02	0.417	0.441	0.429	0.618	IIm
	0	0.008	0	0	0	0.497	bIIM
	0.25	0.01	0.267	0.25	0.258	0.568	bIIm
	0.366	0.028	0.333	0.366	0.349	0.594	IIIM
	0.176	0.009	0.231	0.176	0.2	0.552	IIIm
	0.32	0.014	0.348	0.32	0.333	0.595	bIIIM
	0.385	0.009	0.333	0.385	0.357	0.644	bIIIm
	0.539	0.049	0.529	0.539	0.534	0.685	IVM
	0	0.007	0	0	0	0.497	IVm
	0.613	0.105	0.62	0.613	0.616	0.716	VM
	0.333	0.004	0.333	0.333	0.333	0.59	Vm
	0.408	0.025	0.435	0.408	0.421	0.628	bVM
	0.6	0.005	0.545	0.6	0.571	0.748	bVm
	0.453	0.035	0.393	0.453	0.421	0.652	VIM
	0.265	0.018	0.321	0.265	0.29	0.598	VIm
	0.25	0.005	0.286	0.25	0.267	0.535	bVIM
	0.364	0.006	0.364	0.364	0.364	0.623	bVIm
	0.441	0.025	0.5	0.441	0.468	0.638	VIIM
	0.308	0.01	0.267	0.308	0.286	0.646	VIIm
	0.563	0.015	0.529	0.563	0.545	0.655	bVIIM
	0.25	0.002	0.5	0.25	0.333	0.61	bVIIm
Weighted Avg.	0.5	0.068	0.5	0.5	0.499	0.665	

Ilustración 24: Precisión detallada del ID3

Matriz de Confusión

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	<-- classified as
166	1	3	8	1	0	4	3	4	0	16	2	33	0	1	1	9	4	3	0	0	6	3	0	a = IM
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	b = Im
5	0	16	1	0	0	3	0	1	1	4	0	6	0	1	0	2	2	0	0	0	2	1	0	c = IIM
10	0	0	15	1	1	0	0	0	0	1	0	4	0	1	0	0	0	0	0	0	0	1	0	d = IIm
0	0	0	0	0	2	1	0	0	0	0	0	1	0	1	0	1	0	0	0	2	0	0	0	e = bIIM
0	0	0	2	2	4	1	0	0	0	1	0	0	0	0	0	1	0	0	0	5	0	0	0	f = bIIm
3	0	2	0	0	1	15	0	3	0	1	0	2	0	2	1	3	1	1	0	5	0	1	0	g = IIIM
4	0	1	0	0	0	1	3	0	0	1	0	2	0	1	1	0	1	0	0	0	0	2	0	h = IIIm
5	0	1	0	0	0	4	0	8	0	1	0	2	0	1	0	1	0	0	0	1	0	1	0	i = bIIIM
1	0	0	0	0	0	0	0	1	5	0	0	1	0	2	0	1	0	0	0	1	0	0	1	j = bIIIm
19	0	2	2	0	0	1	0	1	0	55	0	11	2	0	0	3	3	0	0	0	0	3	0	k = IVM
1	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	2	1	0	1	0	l = IVm
37	0	11	5	1	0	4	2	1	0	9	2	147	0	1	0	9	4	0	0	2	2	3	0	m = VM
0	0	0	0	0	0	0	0	0	0	1	0	0	2	1	0	1	0	0	0	1	0	0	0	n = Vm
1	0	0	0	1	0	3	1	1	4	0	0	1	0	20	2	3	1	0	3	7	0	0	1	o = bVM
0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	6	0	0	0	0	1	0	0	0	p = bVm
6	0	2	0	1	0	2	1	0	1	5	0	6	1	3	0	24	0	0	0	0	1	0	0	q = VIM
6	0	1	2	0	0	0	1	0	0	4	0	9	0	0	0	1	9	0	1	0	0	0	0	r = VIm
3	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	s = bVIM
1	0	0	0	0	0	1	0	1	0	0	1	0	0	2	0	0	1	0	4	0	0	0	0	t = bVIm
2	0	0	0	2	5	4	0	1	3	1	1	4	1	6	0	0	1	1	1	26	0	0	0	u = VIIM
2	0	4	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0	0	0	4	0	0	v = VIIm
3	0	0	1	0	0	0	1	1	0	3	1	2	0	0	0	1	1	0	0	0	0	18	0	w = bVIIM
0	0	0	0	0	0	0	1	0	1	0	1	0	0	2	0	1	0	0	0	0	0	0	2	x = bVIIm

Ilustración 25: Matriz de confusión del ID3

AIII.5. Clasificador J48 (C4.5):

Correctly Classified Instances	594	37.8827 %
Incorrectly Classified Instances	974	62.1173 %
Kappa statistic	0.2843	
Mean absolute error	0.057	
Root mean squared error	0.1984	
Relative absolute error	76.9993 %	
Root relative squared error	103.1439 %	
Total Number of Instances	1568	

Precisión detallada por clase

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.611	0.217	0.444	0.611	0.515	0.753	IM
	0	0.001	0	0	0	0.476	Im
	0.203	0.026	0.264	0.203	0.23	0.658	IIM
	0.117	0.027	0.149	0.117	0.131	0.658	IIm
	0	0.013	0	0	0	0.633	bIIM
	0.179	0.017	0.161	0.179	0.169	0.637	bIIm
	0.097	0.029	0.14	0.097	0.115	0.618	IIIM
	0.111	0.011	0.15	0.111	0.128	0.634	IIIm
	0.25	0.024	0.213	0.25	0.23	0.674	bIIIM
	0.235	0.006	0.286	0.235	0.258	0.708	bIIIm
	0.365	0.057	0.379	0.365	0.372	0.72	IVM
	0	0.003	0	0	0	0.456	IVm
	0.531	0.145	0.47	0.531	0.498	0.736	VM
	0	0.001	0	0	0	0.563	Vm
	0.274	0.027	0.333	0.274	0.301	0.746	bVM
	0.5	0.002	0.667	0.5	0.571	0.783	bVm
	0.215	0.025	0.315	0.215	0.256	0.689	VIM
	0.116	0.012	0.217	0.116	0.152	0.65	VIm
	0.111	0.001	0.75	0.111	0.194	0.661	bVIM
	0	0.003	0	0	0	0.584	bVIm
	0.466	0.041	0.402	0.466	0.432	0.743	VIIM
	0.2	0.006	0.308	0.2	0.242	0.691	VIIm
	0.429	0.017	0.49	0.429	0.457	0.746	bVIIM
	0	0.004	0	0	0	0.656	bVIIm
Weighted Avg.	0.379	0.092	0.36	0.379	0.36	0.709	

Ilustración 26: Precisión detallada del J48

Matriz de Confusión

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	<-- classified as
212	1	8	8	1	1	6	6	3	1	25	0	55	1	2	1	2	1	0	0	1	4	7	1	a = IM
3	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	b = Im
16	0	14	2	0	0	3	1	1	0	4	0	22	0	0	0	2	1	0	0	2	0	1	0	c = IIM
22	0	2	7	0	2	2	0	0	0	9	0	7	0	2	0	2	0	0	0	1	0	4	0	d = IIm
1	0	0	1	0	2	1	0	0	1	0	0	2	0	5	0	1	0	0	1	3	0	0	0	e = bIIM
2	0	0	1	3	5	2	0	1	0	2	0	3	0	4	0	2	0	0	0	3	0	0	0	f = bIIm
13	0	3	5	1	3	7	0	4	0	5	0	12	0	1	1	5	1	0	0	10	0	1	0	g = IIIM
10	0	1	0	0	0	0	3	0	0	3	0	5	0	1	0	2	0	0	0	0	1	0	1	h = IIIm
5	0	0	0	0	0	5	1	10	0	2	0	6	0	1	0	0	0	0	0	9	0	0	1	i = bIIIM
0	0	0	0	0	0	2	2	0	4	0	0	2	0	3	0	1	0	0	0	2	0	0	1	j = bIIIm
49	0	2	5	1	0	2	0	1	0	50	0	16	0	0	0	3	6	0	0	0	0	2	0	k = IVM
3	0	0	0	0	0	1	0	0	1	0	0	2	0	0	0	0	0	0	0	1	0	1	0	l = IVm
74	0	16	3	0	0	5	0	6	0	12	1	162	0	3	0	9	5	0	0	5	2	2	0	m = VM
5	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	n = Vm
6	0	1	4	2	4	3	0	3	3	1	0	2	0	20	1	1	1	0	1	16	0	2	2	o = bVM
2	0	0	0	2	0	0	1	0	0	0	0	0	0	0	6	0	0	0	0	1	0	0	0	p = bVm
12	0	3	1	1	3	3	3	1	1	7	0	18	0	4	0	17	1	0	0	3	0	1	0	q = VIM
10	0	2	1	0	1	1	1	1	0	6	0	12	0	1	0	1	5	0	0	0	1	0	0	r = VIm
3	0	0	1	2	3	1	0	5	0	0	0	5	0	0	0	1	0	3	0	0	0	3	0	s = bVIM
2	0	0	0	1	1	1	0	3	2	0	1	0	1	1	0	0	0	0	0	2	0	0	1	t = bVIm
4	0	0	3	5	5	4	0	7	0	2	2	4	0	7	0	1	1	1	1	41	0	0	0	u = VIIM
7	0	0	1	0	1	0	1	0	0	0	0	4	0	0	0	0	1	0	0	0	4	1	0	v = VIIm
14	0	1	3	0	0	0	0	1	0	3	0	4	0	1	0	3	0	0	0	1	1	24	0	w = bVIIM
2	0	0	0	0	0	1	1	0	1	0	0	0	0	4	0	0	0	0	0	0	0	0	0	x = bVIIm

Ilustración 27: Matriz de confusión del J48

AIII.6. Clasificador Naive Bayes:

Correctly Classified Instances	519	33.0995 %
Incorrectly Classified Instances	1049	66.9005 %
Kappa statistic	0.2282	
Mean absolute error	0.0587	
Root mean squared error	0.1995	
Relative absolute error	79.2644 %	
Root relative squared error	103.7219 %	
Total Number of Instances	1568	

Precisión detallada por clase

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.602	0.234	0.422	0.602	0.496	0.783	IM
	0	0.001	0	0	0	0.647	Im
	0.13	0.016	0.273	0.13	0.176	0.691	IIM
	0.033	0.023	0.054	0.033	0.041	0.748	IIm
	0	0.007	0	0	0	0.854	bIIM
	0.179	0.009	0.263	0.179	0.213	0.799	bIIm
	0.056	0.017	0.133	0.056	0.078	0.681	IIIM
	0	0.003	0	0	0	0.664	IIIm
	0.35	0.021	0.304	0.35	0.326	0.831	bIIIM
	0.294	0.009	0.263	0.294	0.278	0.914	bIIIm
	0.299	0.067	0.299	0.299	0.299	0.78	IVM
	0	0.001	0	0	0	0.56	IVm
	0.403	0.162	0.375	0.403	0.389	0.719	VM
	0	0.004	0	0	0	0.59	Vm
	0.205	0.044	0.185	0.205	0.195	0.839	bVM
	0	0.001	0	0	0	0.797	bVm
	0.089	0.025	0.159	0.089	0.114	0.646	VIM
	0.116	0.015	0.179	0.116	0.141	0.672	VIm
	0.185	0.015	0.179	0.185	0.182	0.873	bVIM
	0.188	0.003	0.375	0.188	0.25	0.785	bVIm
	0.545	0.054	0.375	0.545	0.444	0.888	VIIM
	0	0.001	0	0	0	0.673	VIIm
	0.429	0.034	0.32	0.429	0.366	0.819	bVIIM
	0	0.003	0	0	0	0.87	bVIIm
Weighted Avg.	0.331	0.101	0.289	0.331	0.302	0.759	

Ilustración 28: Precisión detallada del Naive Bayes

Matriz de Confusión

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	<-- classified as
209	0	1	7	0	1	0	1	0	3	29	1	69	2	0	1	8	2	0	1	3	0	9	0	a = IM
2	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	1	0	0	0	0	0	0	0	b = Im
16	0	9	1	0	1	2	0	1	1	1	0	27	0	1	0	1	0	0	0	2	2	4	0	c = IIM
24	0	1	2	0	0	2	0	0	0	8	0	9	0	0	0	5	2	2	0	0	0	5	0	d = IIm
1	0	1	1	0	0	0	0	1	0	0	0	0	0	8	0	1	0	2	0	3	0	0	0	e = bIIM
0	0	0	1	0	5	2	0	2	0	2	0	0	0	5	0	1	0	1	1	6	0	1	1	f = bIIm
12	0	2	3	1	3	4	0	4	2	2	0	12	1	3	0	7	1	1	0	13	0	1	0	g = IIIM
12	0	1	2	0	0	0	0	0	0	1	0	8	0	1	0	0	0	0	0	0	0	0	2	h = IIIm
2	0	0	0	0	0	0	0	14	0	1	0	5	0	6	0	0	0	1	1	8	0	2	0	i = bIIIM
0	0	1	2	0	1	0	0	0	5	0	0	0	0	3	0	0	0	0	0	4	0	0	1	j = bIIIm
41	0	1	7	0	0	3	0	0	0	41	0	25	1	0	0	1	5	1	0	0	0	11	0	k = IVM
2	0	0	0	1	0	0	0	0	0	0	0	2	0	1	0	0	0	1	0	1	0	1	0	l = IVm
97	1	9	4	0	2	2	2	5	4	22	0	123	0	1	0	4	7	7	0	8	0	7	0	m = VM
5	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	2	0	0	0	0	0	2	0	n = Vm
7	0	2	0	6	3	4	0	1	0	2	0	2	0	15	0	0	0	4	1	25	0	0	1	o = bVM
5	0	0	1	0	0	0	0	0	0	0	0	3	0	1	0	1	1	0	0	0	0	0	0	p = bVm
19	0	2	2	0	1	3	0	0	2	6	0	16	1	6	0	7	3	2	0	4	0	5	0	q = VIM
15	0	1	1	0	0	1	0	0	1	4	0	10	0	0	0	2	5	0	0	1	0	2	0	r = VIm
1	0	0	1	0	2	1	0	10	0	1	0	0	1	2	0	1	0	5	1	0	0	1	0	s = bVIM
1	0	0	0	1	0	1	0	3	0	0	0	1	0	5	0	1	0	0	3	0	0	0	0	t = bVIm
4	0	0	1	1	0	2	0	4	0	3	0	4	0	18	0	0	2	1	0	48	0	0	0	u = VIIM
9	0	1	0	0	0	2	0	0	1	1	0	5	0	1	0	0	0	0	0	0	0	0	0	v = VIIm
9	0	1	1	0	0	1	0	1	0	12	0	5	0	0	0	1	0	0	0	1	0	24	0	w = bVIIM
2	0	0	0	1	0	0	1	0	0	0	0	0	0	4	0	0	0	0	0	1	0	0	0	x = bVIIm

Ilustración 29: Matriz de confusión del Naive Bayes

AIII.7. Clasificador Perceptrón multicapa:

Correctly Classified Instances	392	25 %
Incorrectly Classified Instances	1176	75%
Kappa statistic	0.1501	
Mean absolute error	0.0659	
Root mean squared error	0.2088	
Relative absolute error	88.8898 %	
Root relative squared error	108.5327 %	
Total Number of Instances	1568	

Precisión detallada por clase

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.398	0.171	0.398	0.398	0.398	0.687	IM
	0	0	0	0	0	0.527	Im
	0.145	0.017	0.278	0.145	0.19	0.604	IIM
	0.167	0.052	0.112	0.167	0.134	0.613	IIm
	0.167	0.025	0.073	0.167	0.102	0.633	bIIM
	0.036	0.054	0.012	0.036	0.018	0.623	bIIm
	0.056	0.016	0.143	0.056	0.08	0.532	IIIM
	0.037	0.005	0.111	0.037	0.056	0.576	IIIm
	0.075	0.005	0.3	0.075	0.12	0.684	bIIIM
	0	0.006	0	0	0	0.523	bIIIm
	0.197	0.023	0.45	0.197	0.274	0.677	IVM
	0	0	0	0	0	0.477	IVm
	0.413	0.21	0.322	0.413	0.362	0.667	VM
	0	0	0	0	0	0.518	Vm
	0.151	0.027	0.216	0.151	0.177	0.685	bVM
	0	0	0	0	0	0.56	bVm
	0.076	0.046	0.081	0.076	0.078	0.562	VIM
	0.023	0.087	0.007	0.023	0.011	0.5	VIm
	0.148	0.044	0.056	0.148	0.081	0.63	bVIM
	0	0.003	0	0	0	0.546	bVIm
	0.375	0.039	0.367	0.375	0.371	0.717	VIIM
	0	0.001	0	0	0	0.5	VIIm
	0.25	0.015	0.378	0.25	0.301	0.672	bVIIM
	0	0	0	0	0	0.47	bVIIm
Weighted Avg.	0.25	0.095	0.273	0.25	0.251	0.643	

Ilustración 30: Precisión detallada del Perceptrón multicapa

Matriz de Confusión

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	<-- classified as
138	0	5	18	5	15	2	3	0	0	15	0	88	0	1	0	10	25	16	1	2	0	3	0	a = IM
2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	b = Im
8	0	10	3	3	3	1	1	0	0	3	0	22	0	1	0	2	5	2	0	2	0	3	0	c = IIM
12	0	0	10	1	6	2	1	0	1	1	0	11	0	1	0	3	8	1	0	2	0	0	0	d = IIm
0	0	0	0	3	1	0	1	0	0	0	0	3	0	3	0	2	2	2	0	1	0	0	0	e = bIIM
0	0	0	1	1	1	3	0	2	1	1	0	4	0	3	0	1	4	1	0	5	0	0	0	f = bIIm
14	0	1	2	3	5	4	0	0	0	0	0	12	0	4	0	6	7	3	1	8	0	2	0	g = IIIM
6	0	0	2	0	1	0	1	0	0	3	0	8	0	0	0	3	2	1	0	0	0	0	0	h = IIIm
2	0	1	2	3	4	0	0	3	0	0	0	8	0	2	0	2	4	5	0	4	0	0	0	i = bIIIM
2	0	0	0	2	1	0	0	0	0	0	0	4	0	3	0	2	2	0	0	1	0	0	0	j = bIIIm
31	0	4	10	1	13	1	0	0	1	27	0	21	0	0	0	2	17	6	0	0	0	3	0	k = IVM
3	0	0	1	0	0	1	0	0	0	0	0	2	0	0	0	0	1	0	0	1	0	0	0	l = IVm
66	0	12	15	1	11	2	0	1	3	3	0	126	0	3	0	17	19	15	1	7	0	3	0	m = VM
3	0	0	2	0	1	0	0	0	0	0	0	2	0	0	0	1	1	0	0	0	0	1	0	n = Vm
5	0	0	5	2	8	3	0	0	1	2	0	9	0	11	0	2	6	2	0	15	0	2	0	o = bVM
3	0	0	2	0	1	0	0	0	0	0	0	3	0	1	0	0	2	0	0	0	0	0	0	p = bVm
16	0	1	4	3	5	2	1	0	1	1	0	23	0	2	0	6	9	2	0	2	0	1	0	q = VIM
13	0	2	3	1	1	2	0	0	0	0	0	11	0	1	0	3	1	2	0	1	1	1	0	r = VIm
3	0	0	1	1	2	0	0	2	1	0	0	5	0	3	0	2	1	4	0	0	0	2	0	s = bVIM
1	0	0	2	0	1	0	1	2	0	0	0	2	0	3	0	1	2	1	0	0	0	0	0	t = bVIm
6	0	0	3	9	4	4	0	0	1	2	0	9	0	5	0	3	6	2	0	33	0	1	0	u = VIIM
5	0	0	2	1	0	1	0	0	0	0	0	5	0	0	0	0	1	0	1	3	0	1	0	v = VIIm
7	0	0	1	0	0	0	0	0	0	2	0	10	0	2	0	6	7	6	0	1	0	14	0	w = bVIIM
1	0	0	0	1	0	0	0	0	0	0	0	2	0	2	0	0	1	0	0	2	0	0	0	x = bVIIm

Ilustración 31: Matriz de confusión del Perceptrón multicapa

AIII.8. Clasificador Multipass SOM:

Correctly Classified Instances	376	23.9796 %
Incorrectly Classified Instances	1192	76.0204 %
Kappa statistic	0.0737	
Mean absolute error	0.0712	
Root mean squared error	0.1921	
Relative absolute error	96.1583 %	
Root relative squared error	99.8886 %	
Total Number of Instances	1568	

Precisión detallada por clase

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.591	0.488	0.256	0.591	0.357	0.627	IM
	0	0	0	0	0	0.523	Im
	0	0.008	0	0	0	0.555	IIM
	0.05	0.002	0.5	0.05	0.091	0.614	IIm
	0	0.001	0	0	0	0.733	bIIM
	0	0	0	0	0	0.552	bIIm
	0.028	0.009	0.125	0.028	0.045	0.623	IIIM
	0	0	0	0	0	0.503	IIIm
	0.05	0.009	0.125	0.05	0.071	0.589	bIIIM
	0	0.001	0	0	0	0.717	bIIIm
	0.073	0.035	0.167	0.073	0.102	0.626	IVM
	0	0	0	0	0	0.499	IVm
	0.354	0.261	0.247	0.354	0.291	0.625	VM
	0	0	0	0	0	0.479	Vm
	0.055	0.019	0.121	0.055	0.075	0.695	bVM
	0	0	0	0	0	0.697	bVm
	0.013	0.009	0.067	0.013	0.021	0.623	VIM
	0	0.001	0	0	0	0.554	VIm
	0	0	0	0	0	0.651	bVIM
	0	0	0	0	0	0.706	bVIm
	0.466	0.082	0.252	0.466	0.327	0.751	VIIM
	0	0.003	0	0	0	0.593	VIIm
	0	0	0	0	0	0.707	bVIIM
	0	0	0	0	0	0.809	bVIIm
Weighted Avg.	0.24	0.169	0.17	0.24	0.175	0.631	

Ilustración 32: Precisión detallada del Multipass SOM

Matriz de Confusión

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	<-- classified as
205	0	3	0	0	0	3	0	6	0	15	0	87	0	7	0	2	1	0	0	14	4	0	0	a = IM
3	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	b = Im
37	0	0	0	0	0	1	0	1	0	5	0	22	0	1	0	0	0	0	0	2	0	0	0	c = IIM
28	0	0	3	0	0	1	0	0	0	7	0	18	0	1	0	0	0	0	0	2	0	0	0	d = IIm
5	0	0	0	0	0	0	0	1	0	0	0	5	0	1	0	0	0	0	0	6	0	0	0	e = bIIM
13	0	1	0	0	0	1	0	1	0	1	0	6	0	2	0	0	0	0	0	3	0	0	0	f = bIIm
32	0	1	0	0	0	2	0	0	0	2	0	23	0	2	0	1	0	0	0	9	0	0	0	g = IIIM
19	0	1	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	1	0	0	0	h = IIIm
16	0	0	0	1	0	2	0	2	0	1	0	11	0	1	0	0	0	0	0	6	0	0	0	i = bIIIM
8	0	0	1	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	3	0	0	0	j = bIIIm
74	0	0	0	0	0	1	0	2	0	10	0	43	0	2	0	1	0	0	0	4	0	0	0	k = IVM
3	0	0	0	0	0	0	0	0	0	0	0	3	0	1	0	0	0	0	0	2	0	0	0	l = IVm
165	0	3	0	0	0	2	0	2	0	9	0	108	0	4	0	3	0	0	0	9	0	0	0	m = VM
5	0	0	1	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	2	0	0	0	n = Vm
30	0	1	1	0	0	0	0	0	0	2	0	13	0	4	0	1	0	0	0	21	0	0	0	o = bVM
11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	p = bVm
42	0	0	0	0	0	3	0	0	0	3	0	22	0	1	0	1	0	0	0	7	0	0	0	q = VIM
28	0	0	0	0	0	0	0	0	0	1	0	11	0	0	0	1	0	0	0	2	0	0	0	r = VIm
11	0	0	0	0	0	0	0	0	1	1	0	9	0	0	0	1	0	0	0	4	0	0	0	s = bVIM
8	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	3	0	0	0	t = bVIm
22	0	1	0	1	0	0	0	1	0	1	0	16	0	3	0	2	0	0	0	41	0	0	0	u = VIIM
14	0	0	0	0	0	0	0	0	0	1	0	4	0	0	0	1	0	0	0	0	0	0	0	v = VIIm
18	0	1	0	0	0	0	0	0	0	1	0	14	0	1	0	1	0	0	0	20	0	0	0	w = bVIIM
4	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	2	0	0	0	x = bVIIm

Ilustración 33: Matriz de confusión del Multipass SOM

AIII.9. Clasificador Multipass LVQ:

Correctly Classified Instances	264	16.8367 %
Incorrectly Classified Instances	1304	83.1633 %
Kappa statistic	0	
Mean absolute error	0.0693	
Root mean squared error	0.2633	
Relative absolute error	93.537 %	
Root relative squared error	136.8628 %	
Total Number of Instances	1568	

Precisión detallada por clase

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.499	0.501	0.22	0.499	0.306	0.499	IM
	0	0	0	0	0	0.5	Im
	0.101	0.099	0.045	0.101	0.062	0.501	IIM
	0	0	0	0	0	0.5	IIm
	0	0	0	0	0	0.5	bIIM
	0	0	0	0	0	0.5	bIIm
	0	0	0	0	0	0.5	IIIM
	0	0	0	0	0	0.5	IIIm
	0	0	0	0	0	0.5	bIIIM
	0	0	0	0	0	0.5	bIIIm
	0.102	0.1	0.089	0.102	0.095	0.501	IVM
	0	0	0	0	0	0.5	IVm
	0.2	0.2	0.195	0.2	0.197	0.5	VM
	0	0	0	0	0	0.5	Vm
	0	0	0	0	0	0.5	bVM
	0	0	0	0	0	0.5	bVm
	0	0	0	0	0	0.5	VIM
	0	0	0	0	0	0.5	VIm
	0	0	0	0	0	0.5	bVIM
	0	0	0	0	0	0.5	bVIm
	0.102	0.1	0.057	0.102	0.073	0.501	VIIM
	0	0	0	0	0	0.5	VIIm
	0	0	0	0	0	0.5	bVIIM
	0	0	0	0	0	0.5	bVIIm
Weighted Avg.	0.168	0.168	0.1	0.168	0.121	0.5	

Ilustración 34: Precisión detallada del Multipass LVQ

Matriz de Confusión

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	<-- classified as
173	0	35	0	0	0	0	0	0	0	35	0	70	0	0	0	0	0	0	0	34	0	0	0	a = IM
3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	b = Im
34	0	7	0	0	0	0	0	0	0	7	0	14	0	0	0	0	0	0	0	7	0	0	0	c = IIM
30	0	6	0	0	0	0	0	0	0	6	0	12	0	0	0	0	0	0	0	6	0	0	0	d = IIm
8	0	2	0	0	0	0	0	0	0	2	0	4	0	0	0	0	0	0	0	2	0	0	0	e = bIIM
14	0	3	0	0	0	0	0	0	0	3	0	6	0	0	0	0	0	0	0	2	0	0	0	f = bIIm
37	0	7	0	0	0	0	0	0	0	7	0	14	0	0	0	0	0	0	0	7	0	0	0	g = IIIM
13	0	3	0	0	0	0	0	0	0	3	0	6	0	0	0	0	0	0	0	2	0	0	0	h = IIIm
20	0	4	0	0	0	0	0	0	0	4	0	8	0	0	0	0	0	0	0	4	0	0	0	i = bIIIM
8	0	2	0	0	0	0	0	0	0	2	0	4	0	0	0	0	0	0	0	1	0	0	0	j = bIIIm
69	0	13	0	0	0	0	0	0	0	14	0	27	0	0	0	0	0	0	0	14	0	0	0	k = IVM
5	0	1	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0	l = IVm
153	0	30	0	0	0	0	0	0	0	30	0	61	0	0	0	0	0	0	0	31	0	0	0	m = VM
6	0	1	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0	0	1	0	0	0	n = Vm
37	0	7	0	0	0	0	0	0	0	7	0	14	0	0	0	0	0	0	0	8	0	0	0	o = bVM
6	0	1	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0	0	2	0	0	0	p = bVm
40	0	8	0	0	0	0	0	0	0	8	0	15	0	0	0	0	0	0	0	8	0	0	0	q = VIM
22	0	4	0	0	0	0	0	0	0	5	0	8	0	0	0	0	0	0	0	4	0	0	0	r = VIm
13	0	3	0	0	0	0	0	0	0	3	0	5	0	0	0	0	0	0	0	3	0	0	0	s = bVIM
9	0	1	0	0	0	0	0	0	0	1	0	3	0	0	0	0	0	0	0	2	0	0	0	t = bVIm
44	0	9	0	0	0	0	0	0	0	8	0	18	0	0	0	0	0	0	0	9	0	0	0	u = VIIM
10	0	2	0	0	0	0	0	0	0	2	0	4	0	0	0	0	0	0	0	2	0	0	0	v = VIIm
27	0	6	0	0	0	0	0	0	0	6	0	11	0	0	0	0	0	0	0	6	0	0	0	w = bVIIM
4	0	1	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0	0	1	0	0	0	x = bVIIm

Ilustración 35: Matriz de confusión del Mulitpass LVQ

ANEXO IV: MATRIZ DE COSTES

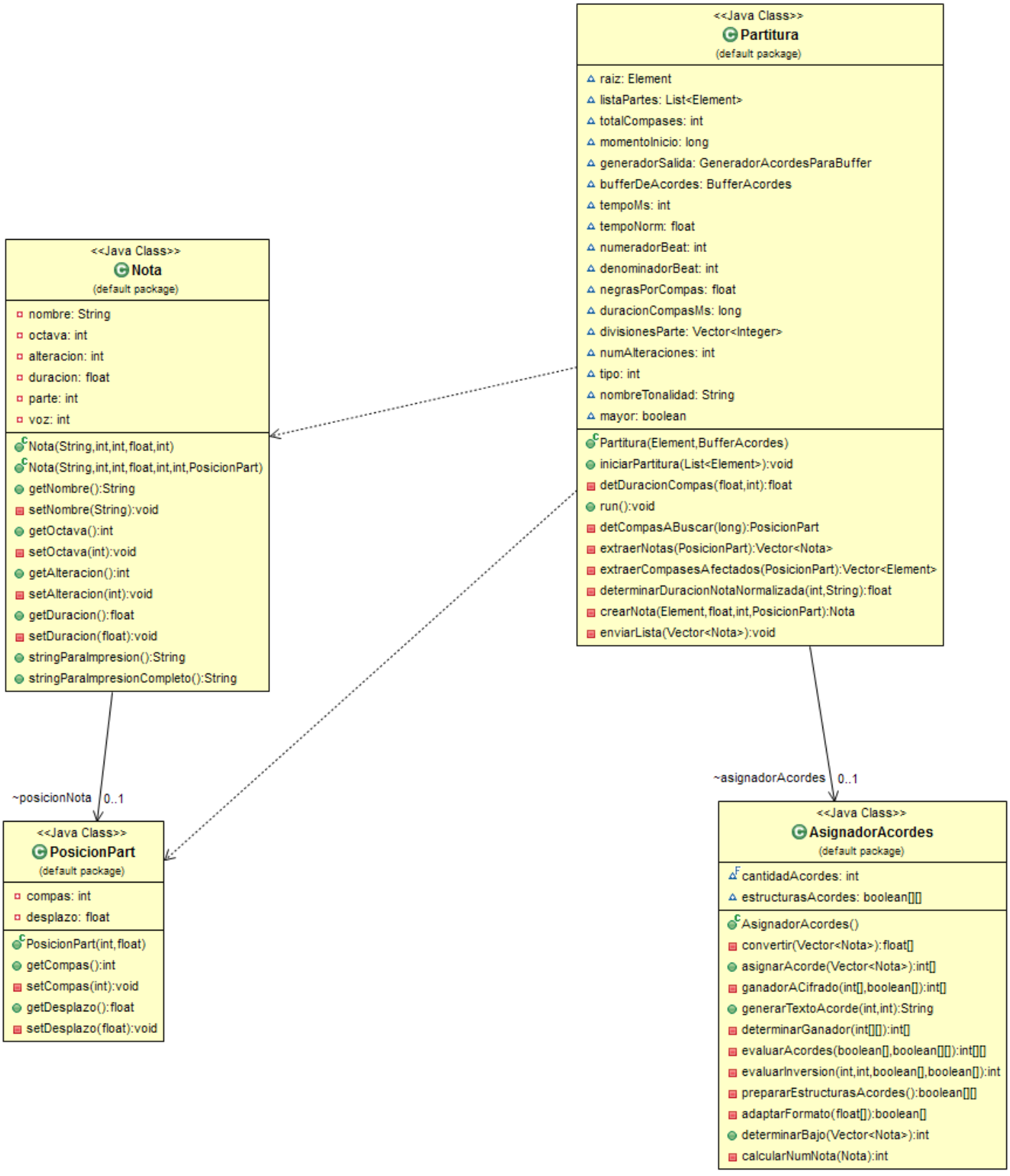
Se empleó la siguiente matriz de costes al evaluar los resultados:

0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0	1.0	1.0	1.0
1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0
1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	0.5	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.75	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.75	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	0.75	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0
0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0
1.0	1.0	1.0	1.0	1.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.75	1.0	1.0	1.0	1.0	1.0	1.0	0.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

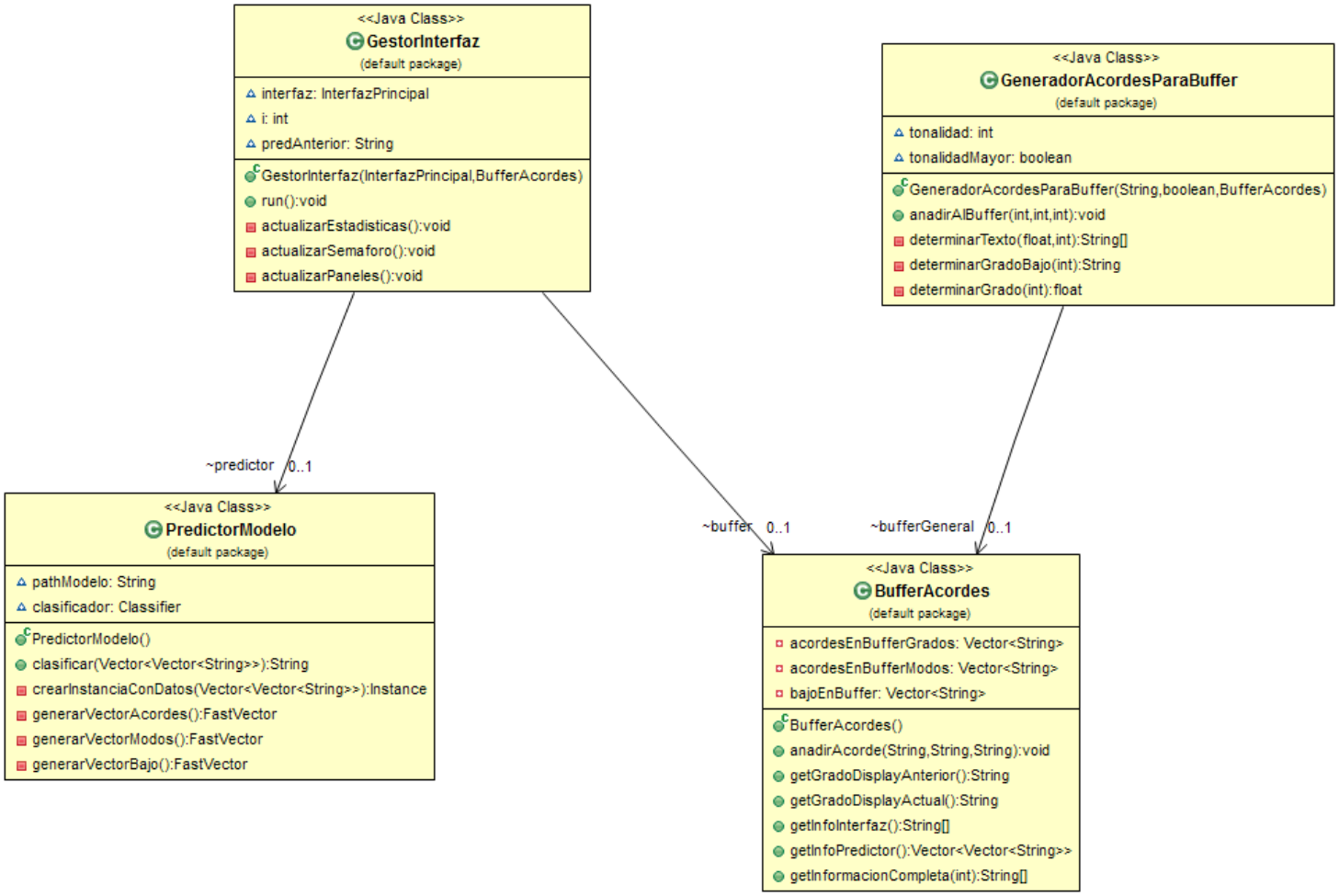
Tabla 5: Matriz de Costes

ANEXO V: DIAGRAMAS DE CLASE

AV.1. Diagrama del parser

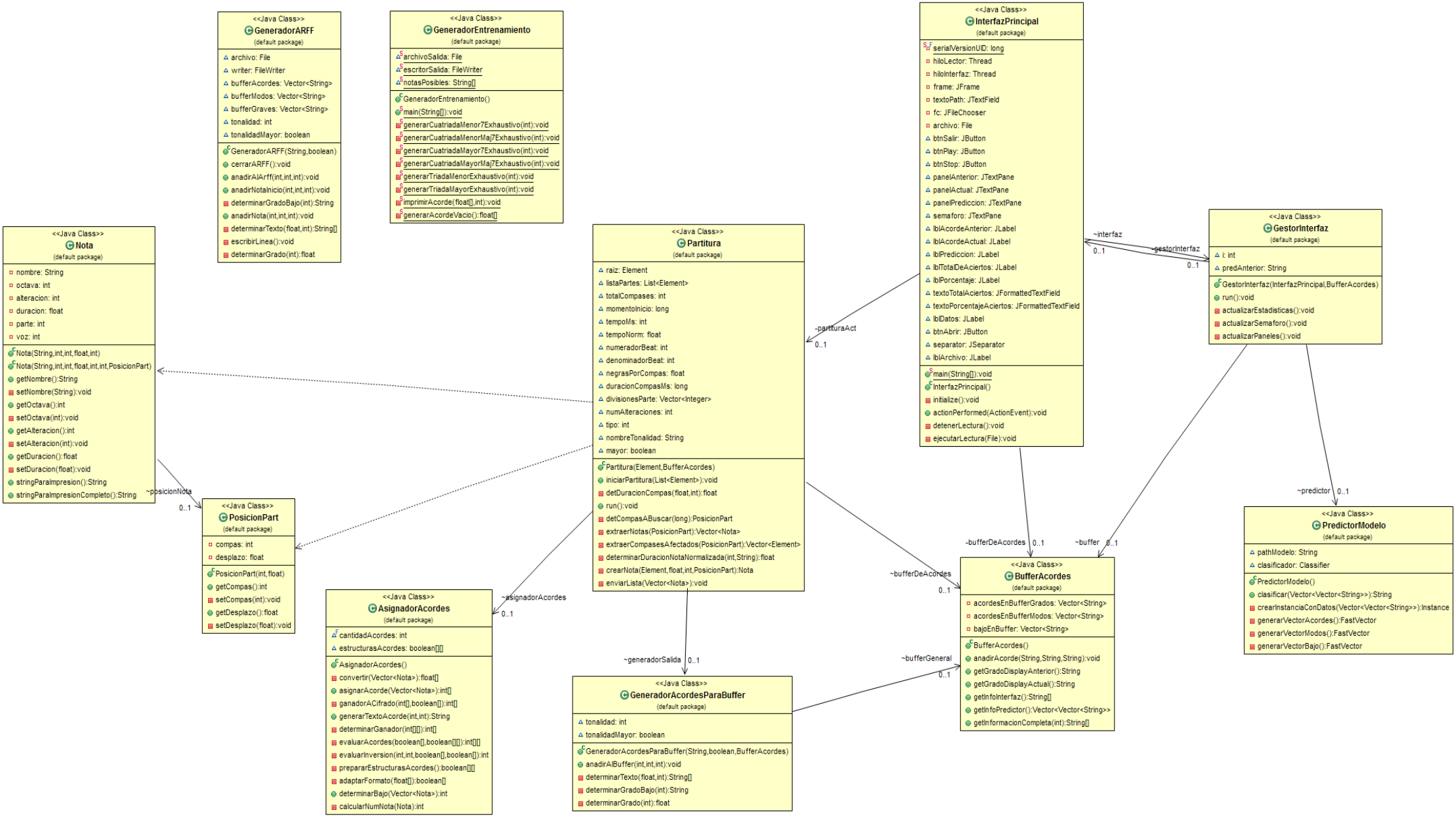


AV.2. Diagrama del predictor



AV.3. Diagrama de la interfaz

AV.4. Diagrama general



SUMMARY IN ENGLISH

Introduction

Mankind has been making music for thousands of years. Since its most ancient beginnings, where musicians only used their voices and bodies to create rhythms and melodies, music has been growing richer and more varied, while musicians have devised increasingly complex structures and motifs. Nonetheless, all occidental music has flourished inside a very definite structure: the rules of harmony. These rules are both inventions and discoveries, and they permeate our musical creation and dictate its development, even of those works which deliberately try to break or avoid them. But to which point do they ensnare composition, and force it to stay in particular paths? Can you predict the evolution of a piece of music from the harmonic framework it generates? Could a machine trained to "listen" to certain works successfully predict what is going to happen in one it doesn't "know"?

The aim of this project is to develop a system able to predict the harmonic development of a piece of music on the basis of the progression that has led to a given point in that work. For this, we will use coral works from the classical period, although the final aim would be a real-time audio analysis. The extraction of relevant features will be done based on expert knowledge, and then entered into a prediction model. Our system will be exclusively probabilistic, and based on training performed with a database.

The project will be divided into several steps:

- Determining which relevant information to extract from the score.
- Developing a reader program which extracts those features.
- Evaluating different prediction models and choosing the most adequate.
- Developing a user interface.
- Implementing a real-time predictor using the components developed up to this point.

Aims

The final aim of this project will be to successfully implement a functional harmonic predictor. To get to that point we will have to reach the following milestones:

Milestone 1: Choose the representation system
The first milestone will be to choose the optimal representation system for this project. Given the circumstances under which this application will be executed and its actual needs in regard to training, the chosen format will be MusicXML.

Milestone 2: Extract information from MusicXML
We will have to implement a file reader capable of extracting all necessary information from the MusicXML score. This will mainly be which notes are sounding in a given moment, but the reader will also have to be able to retrieve information relating to key, tempo, or any other feature which could be of interest.

Milestone 3: Prepare information for processing
Afterwards, we will have to implement a system which transforms the untreated data from the MusicXML score into a file which WEKA can read to train its models. It's at this point in the process where we will abstract information to build a more robust and useful system.

Milestone 4: Create a database
It will be necessary to put together a MusicXML score database big enough to train the models. It will be adapted to our system in regard to instrumentation (coral works) and style (classical period).

Milestone 5: Choose prediction model
The fourth milestone will be successfully creating various WEKA models with the gathered data, in so as to choose the most fitting. The chosen model will be the one the predictor finally uses.

Milestone 6: Build the prediction model into the system
Finally, the model created with WEKA will be added to the application, implementing all the data structures necessary to feed it the required information, and also configuring what the system will do with the generated predictions.

Milestone 7: Implement a user interface
The last step will be creating an interface which allows the user to choose easily which file to analyze, and shows a graphic representation of the results in real time.

State of the Art

The combination of music and computer science has spawned a research field brimming with the most diverse projects and leading to numerous discoveries. These can be divided into three main branches:

Systems for automatic composition:
These are systems which try to create a new musical work. Composition algorithms have existed for hundreds of years, long before any computer was built. One of the most notable examples is the *Musikalisches Würfelspiel*, a game for composing simple tunes with the help of dice devised by Wolfgang Amadeus Mozart.

In any case, the arrival of computer science has expanded our possibilities in algorithmic composition, and in the course of the last 50 years many scientists have invented programs which generate music. The most recent ones tend to apply genetic algorithms to create and evaluate new compositions. A noteworthy example is the *Melomics* sytem developed by the University of Malaga. This computer creates a short piece of music in the style of contemporary classical music in roughly 8 minutes, and adds it to a repository with various millions of others. A selection of this machine's works was recorded on CD by the London Symphony Orchestra.

Intelligent listeners:
These systems take a different approach: they listen to an existing piece of music and work with it to achieve certain goals. These can be, for example, following a score while listening to its performance, or trying to guess which chords are sounding in an audio clip. There are even more ambitious projects: *HarmTrace*, by W. Bas de Haas and his team, is an application which performs a complete harmonic analysis of a score, and has as of today returned amazing results.

Interpretation systems:
Interpretation systems are a field which has been growing stronger in recent years. Its aim is to create a system able to perform music expressively, like a human would. In its beginnings, the implemented programs wrote indications on the music score, thus helping a performing musician. With the rise of MIDI and audio manipulation programs, researchers began to create applications which are able to modify the sounds themselves. The best example for this is the *Director Musices* system, which shows a graphical interface in which you can tinker with diverse parameters to modify the performance of a melody, making it sound happier, or more dramatic.

Our system will be a combination between the first two of the branches described above: At its core, it is an intelligent listener, which receives a score and tries to extract useful information from it. On the other hand, the goal of this extraction is to make a prediction, which, in a way, is a spin on automatic composition.

Development

Our project was divided into two separate parts: Building a predictor and implementing an interface for it. We will discuss each of them separately.

Predictor

Building the predictor was basically a data mining process. As such, we based our workflow on the CRISP-DM methodology (*Cross Industry Standard Process for Data Mining*). We divided the development into six steps:

1. **Understanding and defining the task**
   A prediction model is essentially a classifier: it takes data and classifies them into a particular group, in this case "groups of chords followed by chord X". In our case, there were 24 possible classes (each of the twelve musical degrees, in its major and minor variant). For our classification we would only use information that had appeared before the current moment, since our aim would be real-time execution.

2. **Data collection**
   As the source for our database we used choral works from the late classical period by Haydn, Beethoven and Schubert, since they tend to adhere more strictly to the rules of harmony. Another reason for choosing pieces written for choirs is that these kinds of works tend to have less simultaneous notes (generally only 4) and fewer ornamentations. To read these scores we developed a parser which simulates a performance, and periodically reads which notes are sounding at that moment. This generated enough raw data to model and afterwards train our predictor.

3. **Data modeling**
   Once we had read all simultaneous notes from the scores, we had to model them and extract the most relevant features. In our case, the modeling involved two steps: tagging and abstracting. The first step (tagging) consisted in telling which chord the sounding notes corresponded to. The second step (abstracting) transformed this chord into the corresponding degree (and thus harmonic function) in relation to the musical key of the work. This helped to separate the actual information from the current piece, and made it possible to apply the knowledge gained to works written in other keys. To describe each extracted chord, we settled for three different attributes: **core triad**, **mode** and **bass**. These three separated all information into groups easy to work with. We decided to base predictions on the current chord and the four previous chords, since that offered enough background information to make an informed prediction without taking into account too many variables, which would slow our system down.

4. **Generating and evaluating models**
   After modeling all data we trained different prediction models with it and compared their results. We used 9 different algorithms: Zero R, Decision Table, BFTree, ID3 Tree, J48 Tree, Naïve Bayes, Multilayer Perceptron, Multipass SOM and Multipass LVQ. The model which gave the best results was the J48 Tree, achieving

a success rate of 37.88%. This was 15% above the basic (ZeroR) predictor, which only achieved a success rate of 22.13%. Thus, we chose to use the J48 model in the final implementation.

5. **Deployment**
   In the CRISP-DM methodology, the deployment phase is where the knowledge obtained is put to use. In our case, usage was the subject of the entire second part of the project, i.e.: implementing the final application with a user interface. We will discuss this phase in the next chapter.

Interface

The design of the final user application followed the traditional guidelines of a simplified software development process. As such, it had to specify requisites and use cases which our program would have to fulfill. We set the following requisites and use cases:

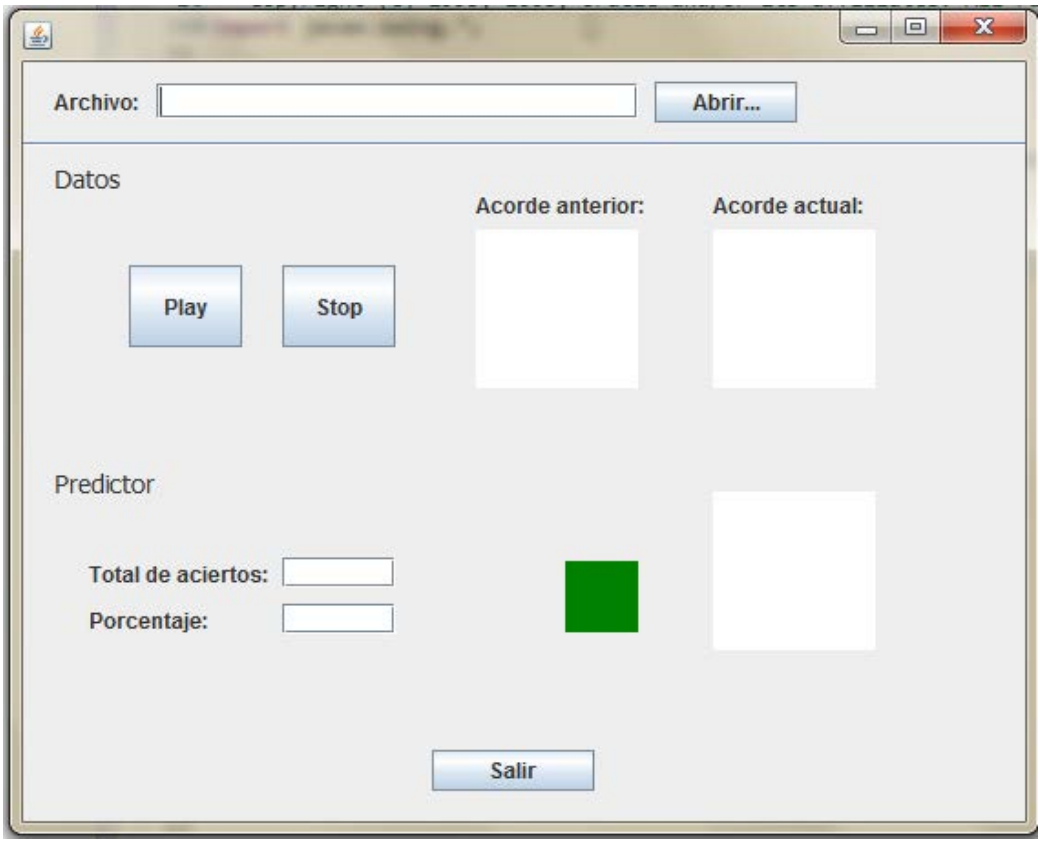
Reference	Name	Description
Functional requisites		
RF-01	Menu	The application will have a graphical interface as main menu.
RF-02	File chooser	The application will allow to select a file through a file chooser
RF-03	Play	The application will have a button to start playback.
RF-04	Stop	The application will have a button to stop playback at any moment
RF-05	Statistics	The application will show all success/error statistics from the current execution.
RF-06	Visual feedback	The application will show the current state of playback and predictions in a clear and intuitive way.
Non-functional requisites		
RNF-01	Portability	The application will have to allow execution on different operating systems.
RNF-02	Usability	The application will have to be usable by persons lacking deep technical knowledge
RNF-03	Executable File	The application will have to be contained in an executable file.

Use Cases		
Reference	**Name**	**Description**
CU-01	Load file	The user chooses upon which file to run the predictions.
CU-02	Start analysis	The user starts the analysis.
CU-03	Stop analysis	The user stops the analysis.
CU-04	Exit	The user exits the application.

While designing the interface, our main goals were to make it simple and intuitive, so we decided to show everything in a single screen and streamline the user options to the most basic interactions. Thus, the screen would show the following elements:

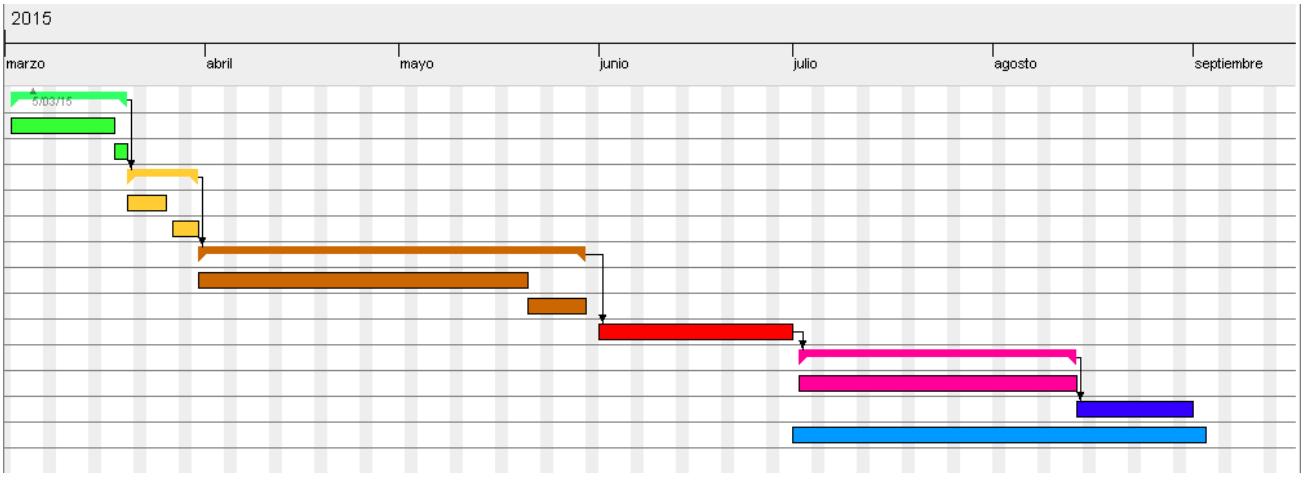
- The name of the file being analyzed.
- Start and Stop buttons.
- The previous chord, the current chord and the prediction.
- A visual indicator which shows quickly if the prediction was right.
- A box with statistics of the current analysis.

We developed the interface using the Java SWING library through the Eclipse plug-in "Eclipse Window builder". For the final version we added an "Exit" button on the bottom of the interface. The final developed interface looked like this:



Planning

To plan and develop our software, we chose the Extreme Programming methodology. Since it is designed for small teams and focused on fast results, it seemed the most adequate for this project. It also has a strong focus on revising results and refactoring code, which fits what is expected from a project of our kind. Though while planning the project we had estimated a total time of 105 work days (with a halt in development during June due to entrance exams in colleges in Germany), the final project suffered some delays, and full development ended taking up 113 work days. The GANNT diagram of the final development schedule finally looks as follows:



Initial study	2/03/15	19/03/15	14
State of the art	2/03/15	17/03/15	12
Define goals	18/03/15	19/03/15	2
Analysis	20/03/15	30/03/15	7
System requisites	20/03/15	25/03/15	4
Use cases	27/03/15	30/03/15	2
Implementation - Part 1	31/03/15	29/05/15	44
Parser	31/03/15	20/05/15	37
Choose classification algorithm	21/05/15	29/05/15	7
Development pause	1/06/15	30/06/15	22
Implementation - Part 2	2/07/15	13/08/15	31
Interface and classifier	2/07/15	13/08/15	31
Tests	14/08/15	31/08/15	12
Documentation	1/07/15	2/09/15	46

Tests

In the course of this project, we tested each part of the application separately until it delivered the results we wanted. These test batteries can be divided into two groups: the entire testing done on the parser, and the tests for the general application.

Tests done on the parser:
While developing the parser, we had to check if the features extracted from the score were correct. This had to be done by hand, printing on the screen which notes and their corresponding chords were being read. We represented each reading the following way:

```
Compas actual: 2 // Desplazo: 0.44666666
Notas actuales: [P0 C5(0.5/1)], [P0 G4(0.25/1)], [P1
E3(0.25/1)], [P1   C3(0.5/1)],
Acorde: C
```

Even though at the beginning we fed the system specially prepared simple scores, we quickly changed to "real" scores. This helped us find a lot of errors and deviations from "standard" MusicXML, which made our system more robust to, for example, rests and triplets. All these things helped perfecting the parser, which led to cleaner data for our prediction model.

Tests to the general application:

In the end phase of the development of the project we had to test how well the final program worked. Most basic functionality tests were also done manually, checking the information moved through the panels in the right way, and that predictions were updated in time. We also ensured that all design requisites set at the beginning of our project had been adequately met, such as an interface with buttons, visual feedback, and a file selector. Finally, we ran our whole database through the application and measured its accuracy. Our test showed an average success rate of 36.03%, slightly below our model tests. This can easily be due to the (slightly) random nature of our parser: while simulating the reading of a score the exact moment of time in which we extract the notes varies from execution to execution and, therefore, it is common that in some instances the readings may be different.

We built our databases exclusively with public domain scores. Since the main repository of these (imslp.org) doesn't have available scores in MusicXML format, we took the scores from hausmusik.ch, an association for the promotion of home musicianship, and cpdl.org, the Choral Public Domain Library. These websites offer extensive libraries of public domain classical music in various formats.

Results and development perspectives

Here we will review the results obtained during the development of our project. We will also discuss possible paths of further development.

Review of the achieved results

The aim of this project was to develop a system able to predict the harmonic progress of a piece of music on the basis of the chords that precede any given moment. We chose a statistic model, and used choral works by Beethoven, Haydn and Schubert to train it. We achieved the following goals:

Development of a parser capable of interpreting a MusicXML sheet: We created a working XML parser prepared for real-time execution. Special care was given to its robustness, which makes it capable of reading scores from different sources which might give less information than what is considered standard.

Extraction of useful information from the score: We applied expert knowledge to choose which information from the score we would use. The parser was designed in a way which allows new criteria to be implemented easily. This was proven when we added the bass notes to the analysis, something missing in the first design. Implementation was quick and only suffered minor delays.

Classification of chords: We designed an expert system which assigns the chord corresponding to the notes which are extracted at a given point in time. Our system is able to recognize 8 different kinds of chords, but is designed in a way that makes it easy to add more.

Putting together of a database to create and train a model: We created a database with a total of 1568 items from 20 different works. Similar bases from other epochs were developed as a reference, but were not used to train the prediction model.

Selection of a prediction algorithm: A total of 9 different algorithms were developed to choose the best and use it for our predictions. The winning algorithm, C4.5, chose one of the 24 different classes with a 15% the success rate over the base predictor.

Development of a user interface: We developed a graphic user interface to use the program, which offers information and allows usage in real-time. It was designed in a way which allows users without technical knowledge to use it easily.

In conclusion: as presented in the design objective, **we have developed a completely functional harmonic predictor.** Even if the results do, for the present, only allow use within the framework of research, they are very promising, and offer a solid and ample groundwork on which to build in the future. We have also proven that predictions made upon information extracted during a discrete time window can be correct. The system doesn't need any information about the work's structure, nor does it need to know its complete development.

At this point, different ways for future development open up. We will discuss them in the next chapter.

Future lines of work

With the current success rate the program is not yet suitable for use in the "real world". On the other hand, due to the approach we have chosen, which is oriented towards a practical application of the program, its interest as an instrument for general research is, to a certain point, limited.

However, this application offers multiple possibilities for future development. They can be divided into two groups: improving the prediction quality, and preparing the application for general usage. These two possible paths, of course, intertwined.

Perfecting the predictor

Even if the results are quite respectable right now, it would be vital to explore up to which point the success rate of the predictor can be increased. Breaking the 50% barrier would be desirable, but any kind of general use would presuppose a rate of at least 70%. This objective could be tackled from different perspectives.

Rethinking which information is extracted from the score: We could search for new indicators of the work's future evolution, modify the number of measures which are used for prediction or introduce new, rarer chords. This is the least promising approach, since we have not found any highly significant but so far overlooked attribute: excess data would slow the program down and too many attributes could pave the way to *overfitting*, even if the intention of this work has been to always try to abstract and generalize.

Modify the way in which the program decides which chord is sounding: We could change the current method, which takes a "photograph" of the moment, to one that collects all notes that sound in a given time frame, weighs their importance according to values such as length or repetitions and, using that information, tries to infer which chord is sounding. How much this could improve our prediction rates is hard to say: in the end it is very difficult to estimate the error rate of the photographic method, since there is no way of knowing if the chord read at that exact moment is perhaps one of the non-representative ones of the measure. Also, the system would be exposed to other risks: widening the measuring to a window creates the risk of mixing two or more consecutive chords, which would lead to erroneous classification. It wouldn't be possible to establish a prediction based on the whole measure, since it is frequent to change chords often within a short period of time, and as such, choosing the exact length of the fragment to analyze would be a dilemma in itself: if it were too long, we could incur the risk just explained, if it were too short, it would become too similar to the "photographic" method.

Implement a modulation detector: While modulation to the minor relative key is represented in our predictor, if the work modulates to a different key, the predictor will keep analyzing it as if it were still in the original, adding noise to our data since the progressions stop making sense. A modulation detector which could identify and "translate" the musical degrees it reads to the new key would not just delete that noise, but make it into useful data. Unfortunately, a system which not only detects a modulation at the moment it happens but is also able to interpret correctly into which of all possible keys it has changed is something very complex to develop. Even when we analyze works manually, modulations are typically detected in hindsight: we search

for a point in which it is completely clear that we are in a new key, and which one it is (generally searching for a V-I cadence), and from there we go back through the score assigning the corresponding degrees. Doing all this in real time would be well worth a project in itself.

Conversion into an application for general use

Once we increase the percentage of correct predictions, we could plan make the application available to the general public. It could be especially useful for jazz and rock musicians. In these fields, "jams" (group improvisations) are common, and our application, if left running during the session, could work as a cheat sheet for a disoriented musician during a structured jam (where harmonic progressions are defined beforehand) or offer suggestions in free jams (where there is no fixed progression). Of course, we would first have to solve the following problems:

Real audio input: We would have to develop a system able to work with real audio input through a microphone. Even if a simple tuner (that is, a microphone with a program which tells if the sounding note has the right pitch) is a simple application and has been a standard in the music industry for several years, identifying various simultaneous notes increases the difficulty exponentially. In the "State of the art" section of the Spanish version we discuss two projects which are trying to do exactly this, with good results. A new approach would be to use the S.P.E.A.R software: since it separates a sound into the harmonics which compose it, we could try to deduce from the ones that sound stronger (and thus are shared by the majority of notes) which chord is sounding, without needing to analyze every single note. The system would work with a "general impression" of the sound atmosphere and work with it. Of course, no matter which method we choose, we would have to see if the analysis could be made in real time.

Port to smartphones: Once we develop a real audio input, the next challenge would be to port the application to smartphones, since the target audience would not necessarily have extensive technical knowledge and would use the application as an accessory, not as an end in itself. In this case, having developed our program in Java will be of great help, since its adaptation to other devices is a lot easier. Nonetheless, it would still be necessary to check if running the application on a system with less processing power would allow for real time use, which in our case is necessary.

Evaluate how the system works with different genres and epochs: We could assess how the system works with other music genres. These would not need to be only the already mentioned Jazz and Rock: if we find a way to detect modulations we could, for example, try to analyze works from the romantic period with some hope of succeeding. In the case it should not work with the desired accuracy, we could try to train the model with an even more modern database. This would bring new problems, since most contemporary music is still under copyright law, and creating a sizeable training database could prove expensive. Also, this kind of music is more difficult to find in traditional score format: usually scores are sold as a tablature or with only chord names, which would require a different parser.

Developing an automatic accompanist: If we manage to successfully complete the aforementioned aims, we could tackle a new challenge: a system that not only predicts which chord is coming, but actually plays it, taking an active part in the performance.

At the beginning, this accompanist would only play the bare chords, but in the future, and after some experimentation, we could try with richer accompaniment methods. It should be noted that our system would only be able to generate accompaniments for a musical ensemble, or an instrument which creates some harmonic fabric by itself. Creating real-time accompaniment for a melody is a completely different research line.

As we see, the implemented system offers numerous research alternatives, and lays the foundation for a path which can lead to thrilling results.